OBJECT-ORIENTED PROGRAMMING

September/October 1990 Volume 8.5

Programmer's Journal

.95 Canada $5.95

TM

Design a Virtual Memory System in C and C++

page 40

Pitfalls in Turbo Pascal OOP

page 52

Write Applications that Travel

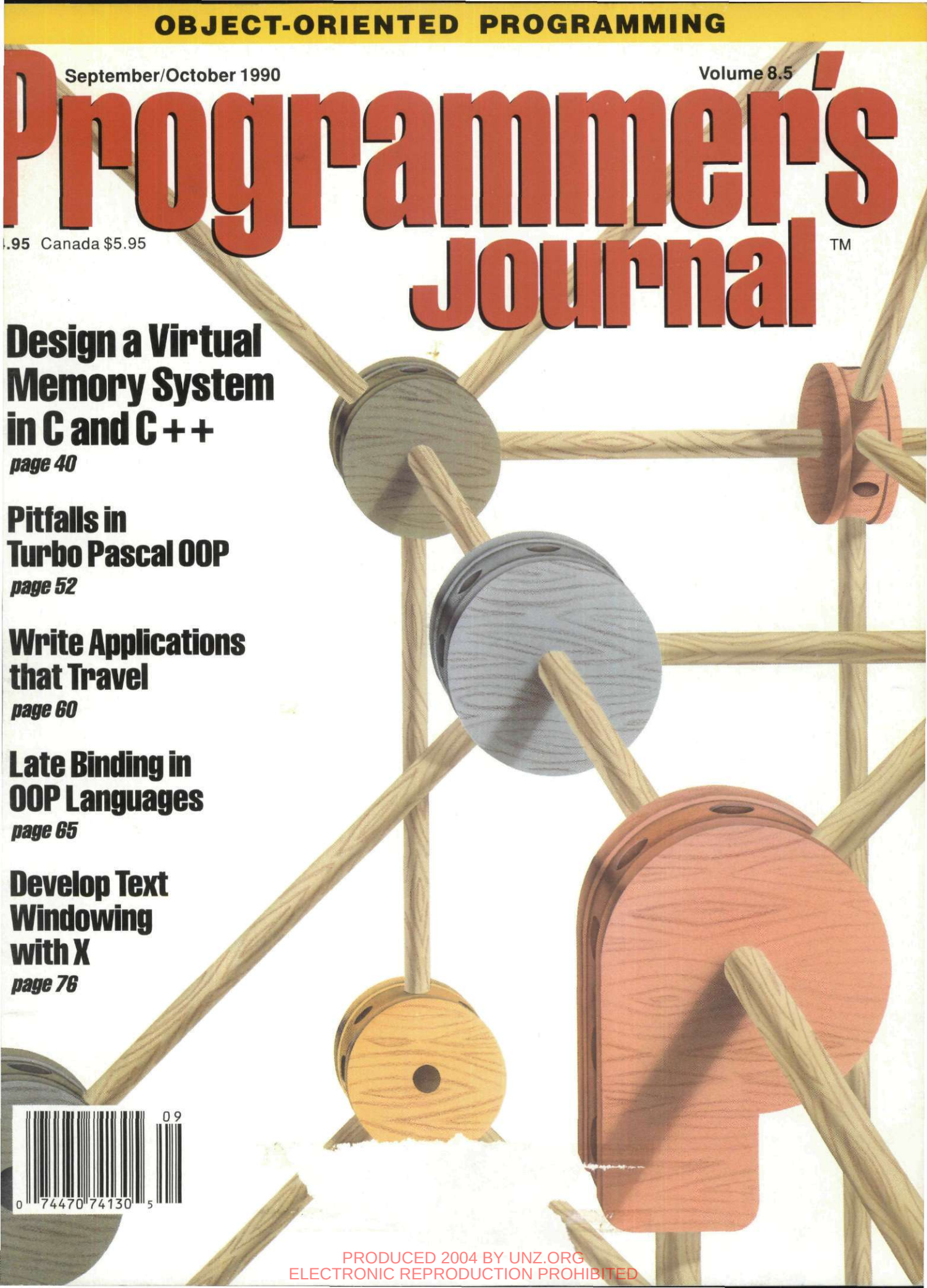
page 60

Late Binding in OOP Languages

page 65

Develop Text Windowing with X

page 76

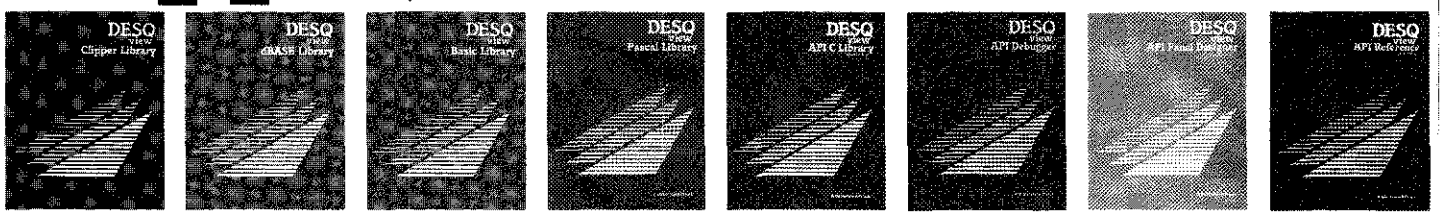


09

0 74470 74130 5

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now you can bring the power of DESQview to Clipper™, dBASE™ and Basic.


DESQ view Clipper Library
DESQ view dBASE Library
DESQ view Basic Library
DESQ view Pascal Library
DESQ view API C Library
DESQ view API Debugger
DESQ view API Panel Designer
DESQ view API Reference

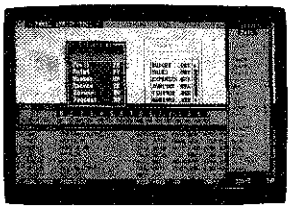
Quarterdeck can provide everything you need to put DESQview capabilities into your applications.

There's a growing realization among programmers that for many users, the ideal PC of the future doesn't look all that different from the PC of today. Experts agree that extended DOS will be the dominant operating system for the foreseeable future.

And DESQview is the best way to create powerful, multitasking solutions for the 30 million DOS PCs in use today.

The API Advantage

Programmers who take advantage of DESQview's API (Application Program Interface) get easy access to the powerful capabilities built into DESQview—multitasking, windowing, intertask communications, mailboxes, shared programs, memory management, data transfer, mousing, menu-building and context-sensitive help. In short, you get all the important benefits promised by OS/2™. Without the overhead burden and performance trade-offs.



Interactive tools like this Panel Designer make implementing the DESQview API a snap.

API capabilities are now available to Clipper, Basic and dBASE users as well as Pascal and C programmers.

Bells and Whistles

A program taking advantage of the DESQview API can spawn subtasks for performing background operations or new processes for loading and running other programs concurrently. It can schedule processing after an interval or at a certain time. It can use DESQview's intertask communications to rapidly exchange data between programs, share common code and data; or interrupt at critical events. It can use DESQview's menuing and mousing capabilities to create menus. And there's lots more it can do.

80386 Power

Programmers can take advantage of the 80386's protected mode for large programs, yet run on DOS and multitask in DESQview—side by side with other 80386 and DOS programs. The breakthroughs that make it possible: DOS Extenders from PharLap, Rational Systems, Eclipse and Oracle and DESQview support of these extenders.

A Complete Interactive Toolkit —Including a Debugger

API Reference Manual The key to the power of the DESQview API, this manual contains all you need to know to write Assembly Language programs that take full advantage of DESQview's capabilities. And there's an 'include' file with symbols and macros to aid you in development.

API Clipper Library Add new power to client-server database applications. Define tasks that monitor or control any other application; run reports, sorts or numerical analyses in the foreground or background; create custom interfaces with control over keyboard, mouse and window functions. And it's all easy to accomplish because you can treat the entire set of over 300 object-oriented API functions as prewritten UDFs.

"With the DESQview API, you can create programs with abilities far beyond those of Clipper alone... obviously, the uses are limited by your imagination"
—Data Based Advisor, December 1989

You get the API Reference Manual and source code for the library with the Clipper Library package.

API dBASE Library All API functions are available within dBASE III and IV. We include source code for the library, the API Reference Manual and example programs.

API Basic Library The Basic library provides interfaces for the entire set of API functions. Included are the API Reference Manual, source code for the library and example programs.

API Pascal Library The Pascal library provides interfaces for all API functions. It supports Borland Turbo Pascal 4.0 and 5.0 compilers and Microsoft Quick Pascal. Included are the API Reference Manual, source code for the library and example programs.

API C Library You get C language interfaces for the entire set of API functions. It supports the Lattice™ C, Metaware™ C, Watcom C, Zortech C, C++, Microsoft® C, and Turbo C compilers for all memory models. Included with the C Library package is the API Reference Manual and source code for the library.

API Debugger The DESQview API Debugger is an interactive tool enabling the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter,registers, and stack at the time of the call. Trace conditions can be set to report only calls of interest.

API Panel Designer This interactive tool helps you design windows, menus, help screens, error messages, and forms. It includes an editor that lets you build an image of your panel using simple commands to enter, edit, copy, and move text, as well as draw lines and boxes. You can then define the characteristics of the window that will contain the panel: its position, size, and title. Finally, you can specify the locations and types of fields in the panel.

The Panel Designer automatically generates all the DESQview API data streams necessary to display and take input from your panel. These data streams may be grouped into panel libraries and stored on disk or as part of your program.

Emerald Bay Developers:
DESQview works with Emerald Bay to allow:
- *Separating the DBMS from the application.*
- *Running up to 16 applications that share data simultaneously*
- *Developing network applications on a single PC*
- *Developing event-driven database applications*
- *Easily porting dBASE and Clipper programs to DESQview/Emerald Bay*

X-Windows Most exciting of all, our X-Windows tools make it easy for you to use this widely accepted graphical interface. Available late Summer.

More Tools are Coming If you'd like to know all the things we're up to in the API Toolkit, call or write for details on our API Conference.

Quarterdeck

Trademarks: Clipper, Nantucket; dBASE, Ashton-Tate; OS/2, IBM; PharLap, Rational Systems, Eclipse, Eclipse; Oracle, Lattice, Metaware, Watcom, Zortech, Microsoft, Turbo C
©1990 Quarterdeck Office Systems

Circle 147 on Reader Service Card.

PRODUCED BY SYLVIA.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer's Journal™

September/October 1990 Volume 8.5

FEATURES

ON GRAPHICS **MORE COLORS ON THE VGA** *By Michael Abrash* 28

Installment 20, in which Michael explores the complexities of VGA color and taps the wisdom of both the Eurythmics and Yogi Bear

OBJECT-ORIENTED **A VIRTUAL MEMORY SYSTEM IN C/C++** *By Allen Holub* 40

Using C and C++ to implement a virtual memory class for data arrays

OBJECT-ORIENTED **PITFALLS IN TURBO PASCAL OOP** *By Tom Swan* 52

Tom documents six pitfalls that trapped him in recent OOP expeditions using Turbo Pascal objects to develop an object-oriented file viewer

OBJECT-ORIENTED **WRITING APPLICATIONS THAT TRAVEL** *By Vaughn Vernon* 60

A useful string resource manager in C to ease writing an application targeted for foreign language versions

OBJECT-ORIENTED **LATE BINDING** *By Brett Glass* 65

Exploring late binding in various object-oriented languages

386 NOW **TOOLS FOR 386 DOS** *By M. Steven Baker* 71

An overview of programming languages available for developing applications using 386 DOS extenders

ASM **WINDOWING TEXT WITH X** *By John Otken* 76

John develops a useful text windowing system based on a handful of X Window Protocol packets

COLUMNS

PJ FORUM **CACHE MEMORIES** *By Hal W. Hardenbergh* 13

CONSULTING **BREAKING THE RULES AGAIN** *By Paul Barkley* 22

OFFRAMP **SPARCS PLOT WORLD TAKEOVER** *By Hal W. Hardenbergh* 96

DEPARTMENTS

EDITOR'S NOTE 4

LETTERS 6

PJ BBS LISTING 67

PJ PRESSROOM 87

PJ BILLBOARD 92

ADVERTISER INDEX 95

COVER

The object-oriented paradigm was conceived to handle the most complex of programming tasks. So whether you're picking up the pieces or just fitting some blocks together, this issue will help you deal with those elusive objects.

Artist: Rob Schuster

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Note that all code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Jay Moore at 800-234-0386, or write to *Programmer's Journal*, P.O. Box 70167, Eugene, OR 97401-0110. *Programmer's Journal* is produced using the following software products: Ventura Publisher,™ Aldus PC PageMaker,® and Corel Draw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company, 150 N. 4th Street, Springfield, OR 97477-5454. Second-class postage paid at Springfield, OR, and additional mailing offices. USPS # 000293. **POSTMASTER:** Send address changes to *Programmer's Journal*, PO Box 70387, Eugene, OR 97401-0121.

© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BORLAND INTRODUCES TURBO PASCAL 5.5 WITH OBJECTS

Borland International is sponsoring a worldwide series of comprehensive, one-day seminars on the benefits of object-oriented technology. For more information call 1-800-342-TURBO.

Be Objective.

Turbo Pascal,® the world-standard Pascal compiler, adds Object-Oriented Programming with our version 5.5. We combined the simplicity of Apple's Object Pascal language with the power and efficiency of C++ to create Turbo Pascal 5.5, the object-oriented programming language for the rest of us.

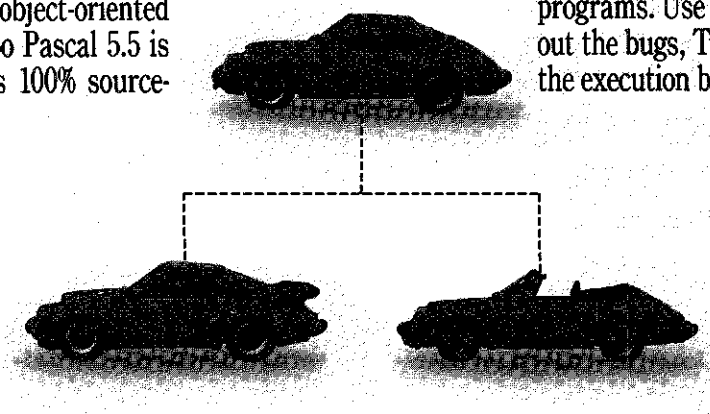
It's easy to extend yourself

If you're already programming with Turbo Pascal, it's easy to extend yourself from structured programming to object-oriented programming. And, Turbo Pascal 5.5 is the *only* compiler that is 100% source-code compatible with your existing Turbo Pascal 4.0 and 5.0 programs.

A fast object lesson

Object-oriented application programs more closely model the way you think. Objects contain both data and code. As in a spreadsheet cell, the value and the formula are together. Objects can *inherit* properties from other objects. For example, a Porsche Carrera inherits most attributes from the base model 911, but it also sports a whale tail.

Turbo Pascal 5.5's object-oriented extensions give you code that's easier to change, extend and support.



Inheritance provides powerful modeling capabilities by allowing objects to inherit attributes from other objects.

Support your objective

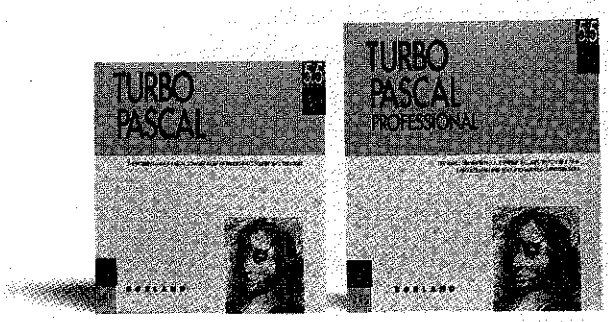
The Turbo Pascal® 5.5 Professional 2nd edition comes with the new Turbo Debugger® & Tools 2.0, which supports building faster, more reliable programs. Use Turbo Debugger to shake out the bugs, Turbo Profiler™ to pinpoint the execution bottlenecks, and Turbo Assembler® to turbo-charge time-critical sections of your program.

Get object-oriented now!

Turbo Pascal 5.5 and the Turbo Pascal 5.5 Professional 2nd edition are available *Now* at the dealer nearest you. Special upgrade prices are available to Turbo Pascal owners.* CALL NOW (800) 331-0877.

Turbo Pascal 5.5
- Compiles @ > 34,000 lines/minute
- New integrated environment tutorial
- Hypertext Help with copy and paste
- Support for 8087/80287/80387

Turbo Debugger & Tools
- Turbo Debugger 2.0
  - Object hierarchy browser and inspectors
  - Reverse excecution
- New Turbo Profiler
- Improved Turbo Assembler 2.0



BORLAND

Code: MA74

*Mail upgrade orders to: Borland, P.O. Box 660001, Scotts Valley, CA 95067-0001. For orders outside the U.S., call (408) 438-5300.

Turbo Pascal, Turbo Debugger, Turbo Profiler and Turbo Assembler are trademarks or registered trademarks of Borland International, Inc. Copyright © 1990, Borland International, Inc. All right reserved. BI 1368A

Circle 49 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

POWER DEBUGGING

BOUNDS-CHECKER

Finds out-of-bounds memory accesses — AUTOMATICALLY.

Flush out those Nasty pointer problems and other out-of-bounds memory accesses — AUTOMATICALLY.

Each time you make a change to a program, run BOUNDS-CHECKER while testing the new code. If you accidentally access out-of-bounds memory, BOUNDS-CHECKER will pop up displaying the offending SOURCE LINE. And your program runs at full speed.

"BOUNDS-CHECKER and Soft-ICE make sophisticated use of the most powerful versions of Intel's processor family to track down some of DOS programming's most insidious bugs. If you're developing programs for DOS, these are essential tools."
PC Magazine
July, 1990 pg. 48

Ship Bug-Free Products

You can run BOUNDS-CHECKER while testing your program. There are no additional steps to your testing cycle, but you can feel secure when the program has passed through BOUNDS-CHECKER with no reported problems.

Many over-write problems and other out-of-bounds memory accesses do NOT show up during normal testing. An out-of-bounds memory location may be modified, but that particular location doesn't happen to be important at the time. Once the program is in the field and a certain network is loaded or a certain T&SR or device driver is loaded, that memory location suddenly becomes very important... AND THE SYSTEM CRASHES.

You can prevent these problems by making BOUNDS-CHECKER a standard part of your testing procedure.

Gives you the protection of a protected operating system under MS-DOS.

BOUNDS-CHECKER uses the 80386 virtual machine technology to provide real-time memory protection. In addition BOUNDS-CHECKER uses the symbolic information output by your compiler to differentiate CODE and DATA. When your program is running, BOUNDS-CHECKER protects the program's CODE and all memory outside your program.

Requires 80386 PC.
MS-DOS is a trademark of Microsoft Corporation.

Soft-ICE 2.5
New Version, New Features

The only debugger specifically designed to solve those problems unique to MS-DOS that we call the DOS Nasties.

- Memory over-writes
- Hung programs
- Program too big to debug
- Debugging T&SRs and Loadable Drivers
- Multiple Symbol Tables
- Supports Microsoft C 6.0 & Turbo C++

MagicCV 3.0
New for C6.0

A set of tools designed to ease the memory crunch with Microsoft C 6.0.

- Run Code View in ***Less Than 8k***
- Run CodeView with EMM & VCPI
- Increase heap space when compiling
- Increase memory with make
- Load high T&SR's and device drivers
- VCPI support

BOUNDS-CHECKER	***$249***
Soft-ICE 2.5	***$386***
MagicCV 3.0	***$199***

Special Offer...

Buy BC & S-ICE	***Save $100***
Buy S-ICE & MCV	***Save $86***
Buy all three	***Save $186***

30 Day Money-Back Guarantee

Nu-Mega TECHNOLOGIES

CALL TODAY (603) 888-2386 or FAX (603) 888-2465
P.O. BOX 7607 ■ NASHUA, NH ■ 03060-7607 ■ U.S.A.

Circle 160 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR'S NOTE

This morning I rose early for a quiet swim in the river before work. The joys of summer are simple pleasures, but the "simple" task of programming appears to be a thing of the past. In this issue of *PJ* we focus on object-oriented programming (OOP), which promises to simplify the programmer's task by hiding some of the complexity.

To this end, new languages have been created and old ones have been enhanced. Like spoken languages, programming tongues evolve to meet changing needs. K&R C has become ANSI C, with C++ nipping at the heels of another ANSI committee. Borland's Turbo Pascal with objects is a far cry from the original Z80 CP/M version. Over time, the better features in one language are often added to others. If the object-oriented paradigm is successful, we'll soon be seeing classes and methods popping up in Object BASIC (soon from Microsoft), COBOL-94, FORTRAN 95X, etc.

Ah, but all paths have pitfalls. First we have the OOP learning curve. Though C++ may be an easy transition for C programmers, ANSI C and AT&T's C++ 2.0 have some serious compatibility conflicts. Other object-oriented languages, such as Smalltalk, Eiffel, and Object Pascal are not available across all platforms. As objects are added to existing languages like Pascal and BASIC, what standards will exist to facilitate cross-platform development? Remember that OOP is designed to handle large, complex programming tasks—applications that would find a ready market on other hardware.

One thing is clear: the days of the C++ translator are numbered. Although CFront (the AT&T C++ translator) and its derivatives continue to be useful products, the native code C++ compilers are coming on fast and furious. Zortech and Borland Turbo C++ were first on the PC scene. JPI, MetaWare, Watcom, and Microsoft will soon have C++ offerings. MicroWay has released a native C++ for Extended DOS and 386 Unix. Oregon Software, the Free Software Foundation (GNU C++), and other vendors have C++ native compilers for several platforms. Besides their faster compilation speed, the native compilers also provide symbolic debugging that is cognizant of the C++ language.

DEALING WITH COMPLEXITY

Large, complex programs appear to be the norm these days—hence the need for OOP. The PC database environments justify this observation. Fox Software has delivered two "free upgrades" of FoxPro since the initial release last November. Ashton-Tate is finally about to provide one to dBASE IV from the initial 1.0 release of over a year ago. A "free upgrade" is really a euphemism for bug fixes to the original product. But these companies don't make profits by providing free upgrades. Obviously some new approaches to programming on large, complex projects are warranted.

The object-oriented paradigm is most often promoted to handle the user interface in graphical environments. But a programmer can easily be bewildered about which future markets to target. Although Windows 3.0 is now the rage, will a new, leaner version of OS/2 or a 386 OS/2 version soon steal the fanfare? These graphical environments extort their price in sluggish performance except when used on higher-end 386 machines. What about all those lowly PCs that run plain DOS applications? On these older machines, text windowing and multitasking with, say, DESQview works remarkably well. Although programming with objects may handle complexity, it often does so at the expense of execution speed and program size.

Today's programmer faces a splintered market and a confusing array of choices. For my part, as long as the weather holds, I'll just part the morning waters and watch the future unravel.

Steven Baker

Programmer's Journal

Editor
M. Steven Baker
Technical Editor
Colin Elligsen
Associate Editors
Cecelia Hagen
Deborah Healey
Contributing Editors
Michael Abrash
Frank D. Greco
Tom Swan

Publisher
Elizabeth Oakley
Associate Publisher
Bobbi Sinyard
Accounting
Jill Edwards
Circulation
Barbara Curtright
Subscriptions
Pamela Coleman
Office Manager
Sylvia Long
Administrative Assistant
Randy Randolph
Sales & Marketing Director
Robert M. Tappan
Advertising Manager
Jay Moore
Production & Art Director
Wanda Walker
Production Artist
Nicole Harris
Illustrations
Jim Carpenter

Business and Editorial Offices
Programmer's Journal
P.O. Box 70167
Eugene, OR 97401-0110
(503) 747-0800
(800) 234-0386
Fax: (503) 746-0071
BBS: See page 67

Subscriptions
1 year (6 issues) $24.00

Change of Address
Programmer's Journal
PO Box 70387
Eugene, OR 97401-0121

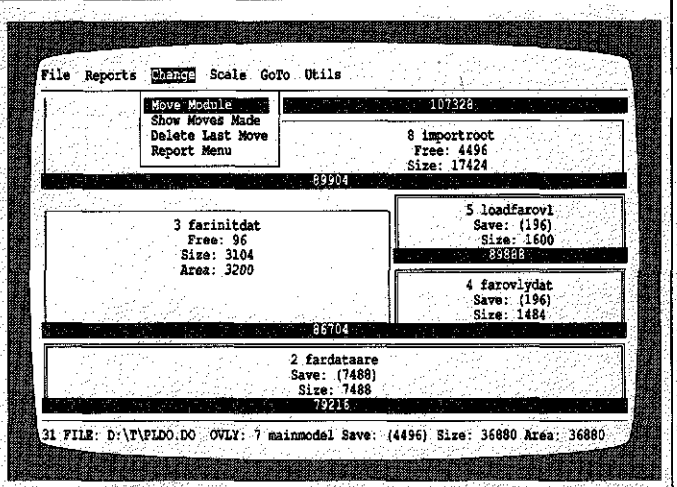
© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OVERLAY DESIGNER™

for Plink86

and RTLink



What You See Is What You Linked

Save Programmers' Time

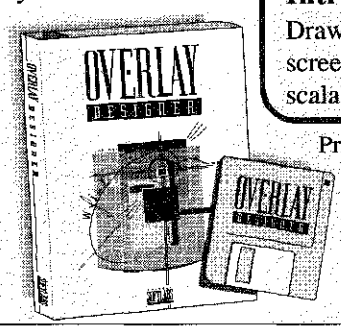
You can expect to save up to 75% of the time you now spend analyzing and testing overlay structures, and still do a better job. Overlay Designer is a screen-oriented data base and presentation program to help programmers design and analyze the overlay structures of large DOS application programs. Overlay Designer ***automatically reads the .MAP file, .OBJ files, .LIB files***, and (for RTLink only) the monitor output captured to file during the link, and builds a database describing the link. The programmer then uses Overlay Designer's graphic, algorithmic, and tabulation tools to design a compact, fast, and reliable overlay structure. Overlay Designer is ***Computer Aided Design*** for DOS overlays. Overlay Designer has been beta tested for 9 months on products with sales from one million dollars per year to one million dollars per month.

Fast Results - Almost No Setup

You can select 3 types of data to import into the Overlay Designer file: 2-dimension overlay diagram, detailed memory sizes, and symbol references. The diagram import takes only a few seconds. One beta test programming team posted a 3 ft x 4 ft overlay diagram on the wall each Friday. The full import takes less time than the link.

A Fully Professional Software Package

Overlay Designer has over 120,000 lines of source code, developed by the primary author of Q&A (® Symantec Corp., InfoWorld Product of the Year 1987, 1988, PC Magazine Technical Excellence Award 1988), and author of Spellguard (® Computer Associates, Infoworld Product of the Year 1981). It is fully documented with an indexed 110 page manual and has context sensitive help at every input screen. Many early adopters ***use Overlay Designer regularly*** and find it an ***indispensible tool*** for overlay design. Overlay Designer is recommended for links with more than 25 overlays, where products that claim to design overlays automatically prove ineffective and simulated virtual memory is too inefficient.



COLEMAN SOFTLABS INC

Coleman Softlabs, Inc.
296 Bay Road,
Atherton, CA 94027
Phone (415) 322-9006
Fax (415) 322-1622

Benefits

- Programmer time savings give ***quick return on investment***
- Diagrams expedite communications in multi-person projects
- Improve morale by ***eliminating tedious detail work***
- Meet project deadlines by reducing overlay design time
- ***Reduce QC testing***; many overlay bugs found automatically
- Improve your product with compact, reliable overlay
- ***Automatic technical documention*** of overlays

"After my first day with Overlay Designer, I had done work that would have otherwise taken me 2 weeks."
- Mike Milsner, Senior Programmer, Symantec Corp.

Features

- Detection of direct ***"equal level" overlay*** calls
- 35 dynamic tables to cross reference symbols & their use
- Separation of references into the categories ***code*** and ***data***
- Scalable screen and hard copy diagram of overlay structure
- Laser printer overlay diagrams ***scaled to memory sizes***
- Free area sizes and potential memory reductions on diagram
- ***"What if" testing of moving memory*** between overlays
- Sort operations on tables to identify important relationships
- Filter operations on tables to eliminate extraneous data
- ***Run editor program*** without leaving Overlay Designer
- Memory size reports by segdef, module, file, & overlay
- Memory sizes reported in hexidecimal or decimal
- ***Stack report tables*** up to 12 deep
- Immediate hard copy of all or part of any table
- Cross-reference publics & externals by file, module, overlay

Introductory Offer: ***Overlay Diagrammer.***™
Draws and labels overlay diagrams instantly - screen and hard copy. Laser printer diagrams are scalable from 3 inches x 3 inches to 7 feet x 7 feet.

Pricing: Includes 60 day warranty & 1 year support.

Overlay Designer:	1 user license $595
	4 user license $1,190
Overlay Diagrammer:	1 user license $149
(90% credited toward Overlay Designer purchase in 3 months.)	4 user license $298

Plink86 ® Sage Software. **RTLink**™ Pocket Soft, Inc.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

WRONG ACCESS?

The article "Accessing high memory" (May/June '90) had some interesting information. However, the programming techniques in this article should not be endorsed. While techniques such as trapping interrupts 15h and 19h and altering memory locations inside other programs may be necessary on rare occasions, they should never be standard practice.

VDISK, circa DOS 3.10, was one of the first programs to use extended memory under DOS. At that time, no conventions for extended memory usage under DOS existed. Kludges were necessary then, but not today.

Memory kludges could be avoided by installing and using the DOS driver HIMEM.SYS, a Microsoft- and Intel-approved method of accessing extended memory. It manages the GateA20 line, high memory (100000h-110000h), extended memory (above 110000h), and even upper memory (0a0000h-100000h). By using HIMEM.SYS, DOS programs, device drivers, and TSRs can use these resources without stepping on each other.

The techniques suggested in the article are likely to break HIMEM.SYS and programs which use it.

Earl Hinrichs
PC Tech, Inc
Lake City, MN

SALFORD RESPONDS TO BAKER

Thanks for the review of DOS FORTRAN compilers for the 386. It was certainly much fairer than some we've seen! Overall, we're very pleased with it, but we would like to bring a few small points to your attention.

For some of the other compilers, mention is made of extensions, but not for OTG FTN77/386. This might be taken to mean that it doesn't have any. To be specific, DO-ENDDO, DO WHILE-ENDDO, and extra intrinsics are mentioned elsewhere; all of these are included in FTN77, and many more as well.

No mention is made of our compilation diagnostics, which we are often told are the best in the business. We have had some users expressing an interest in a portable version of the front end of the

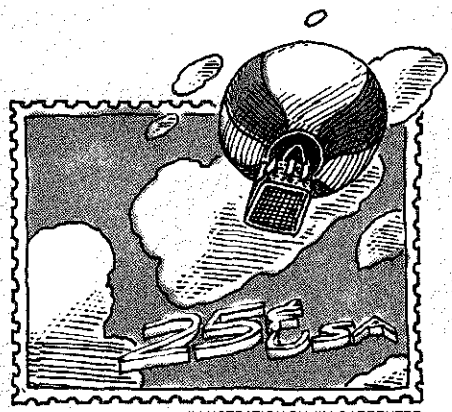


ILLUSTRATION BY JIM CARPENTER

compiler for the diagnostics alone! Of course, you are entitled to disagree.

Much is made of I/O performance, and I am not sure why you focus on buffer sizes as the primary explanation for differing performance. The way we do I/O is by a buffering scheme which avoids switching to real mode DOS for every I/O operation. It may well be a feature of the algorithm that we use that a particular case runs slowly, rather than a direct result of the buffer size. Like many things in computing (hashing, paging, scheduling, etc.) it is possible to construct particular examples which play on an algorithm's weakness. Also in the case of formatted numeric I/O, it is possible to read numbers more quickly if you do it less accurately.

There seems to be no mention of whether any optimization was employed for the benchmarks. The comparative results are marginally worse from our point of view than the results we obtained here for a range of benchmarks. I wonder if some difference in the area of optimization could explain this. Another possible explanation for the discrepancy is that our default INTEGER length is 2 bytes for historical reasons. We intend to add configuration of this option together with the LOGICAL default to our installation program. If you haven't used the /INTL and /LOGL options you would get the short defaults, and short integer instructions are intrinsically slower than long integer instructions, at least partly because they require an operand size override prefix.

DBOS is not a play on Big DOS. Somewhat in the UNIX awk tradition, it is made up from the initials of its designer, together with the letters OS. It was originally an internal name, but we couldn't think of anything that we liked better before release.

Thank you again for the review. We felt it pointed out the strengths and weaknesses of each of the products well.

Tony Webster
Salford Software Marketing, Ltd.
Salford, United Kingdom

THE EDITOR RESPONDS: I'm glad that you felt my review was fair. I try my best to be as unbiased as possible when comparing software products. I must apologize for omitting that the OTG (Salford) FORTRAN compiler supports VAX and other extensions. In fact, all of the 386 DOS FORTRAN compilers now support many VAX extensions including DO-ENDDO, DO WHILE-ENDDO, and additional intrinsic functions.

In my review, I expressed concern about the speed of numerics operations and file I/O. Number crunching is FORTRAN's forte, so most of the programs I used for benchmarking tested the execution of numerics code generated by the compiler (and included in the FORTRAN runtime libraries). My own experience indicates that although a fast 80386/387 or 80386/Weitek combination is nominally 2-4 times faster than an older VAX 785 for number crunching tests, many real applications end up executing slower on the 386 than on the VAX. The difference in performance is most often the speed of file I/O.

Older FORTRAN programs often write out large temporary files to disk. If the DOS FORTRAN compilers and runtimes are not sensitive to the limits of DOS file I/O, the 387 or Weitek number crunching speed advantage can be lost to I/O execution. The default DOS buffer size for file I/O is 512 bytes. Without some compiler option to buffer file input and output, reading and writing to disk under DOS in such small quantities is indeed very slow. On a typical PC hard disk, a whole track can be read in almost the same time as a single sector. The benchmark times for the one file I/O test I included indicate this problem well. The typical 386 FORTRAN compiler generated file I/O code that was about 3 times slower than Microsoft

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

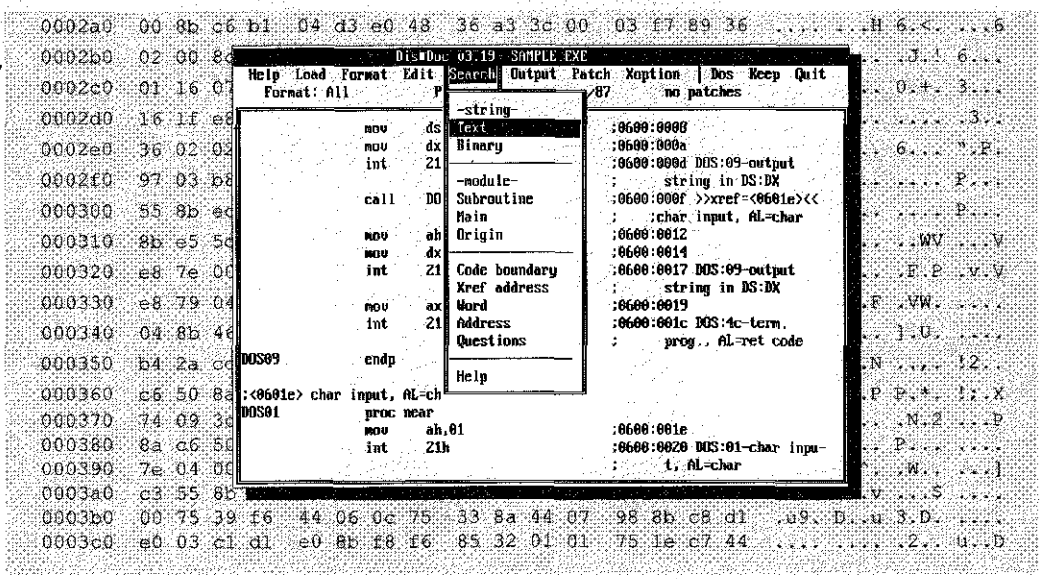
NEW VERSION

No Source?
No Problem.

Automatic BIOS, PSP, MS-DOS interrupt labeling built-in

Batch and Interactive modes

Interactively locates code/data boundaries



8086 to 80486 instruction sets supported

Interactive cross-referencing

Creates listing only after disassembly is perfect

Dis•Doc Professional™

New **Dis•Doc** *Professional* is your **dual-mode** key to any DOS source code. It works in batch and interactive modes simultaneously, allowing you to generate the core information of even the most complex programs **fast**...and modify them even faster. Most programs will come apart in just two minutes. Imagine what you can do with a tool this powerful! **Dis•Doc** sifts through programs **eight times** for guaranteed accuracy. When code gets mixed up with data, our toolbox comes to the rescue with **smart** search and **easy** edit utilities.

Warning: Dis•Doc *Professional* may change the way you work **forever**.

Programmers who used to shy away from fixing outmoded programs with no source code are going to discover a **valuable** new talent: the ability to modify and revise codes that would cost way too much to start over (it's a programming manager's dream). Save your employer huge new-programming fees and enhance your marketability. **Dis•Doc** is so easy to learn, you'll be a high-dollar hero in no time!

Dis•Doc *Professional* is an amazing new teaching tool. Learn how programs work...take them apart and see the writing techniques that top pros use. Use it to assist in debugging. Hunt down viruses and write killers. **Dis•Doc** can save you **years** of frustration, and it only costs $249.95.

COMPARE FOR YOURSELF!

	Dis•Doc™	Sourcer™
Speed of disassembly (bytes/sec)	***822***	*120*
Accuracy (code/data separation*)	***99.9%***	*99.9%*
Unlimited file size	***YES***	*NO*
BIOS labeling	***YES***	*XTRA $$*
BIOS/PSP file labeling	***YES***	*NO*
Unpacks EXE files	***YES***	*XTRA $$*
Patcher	***YES***	*NO*
Batch mode	***YES***	*YES*
Interactive mode	***YES***	*NO*
Time it takes to customize a label	***5 sec.***	*2-5 min.*
Time it takes to change data to code	***5 sec.***	*2-5 min.*

*at eight passes based on a 9040 byte EXE file picked at random

To order your **Dis•Doc Professional**™ disassembler toolkit or our $3 demo disk, simply call:

1-800-336-1961 in US
1-203-953-0236 outside US
or send check or money order for **$249.95** plus $5 s&h to:

RJSwantek, Inc.
178 Brookside Road
Newington, CT 06111-1320 USA

For more information call 1-203-953-0236 • FAX available • MasterCard & Visa • Shipped Immediately Via UPS Blue inside US.

Sourcer is a registered trademark of V Communications, Inc.

Circle 59 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

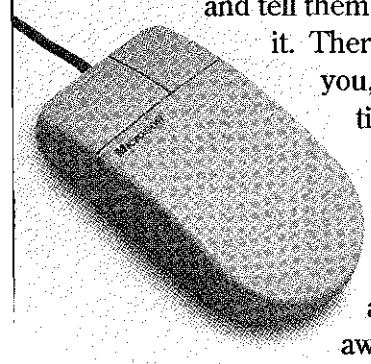
This team created your next generation editor. Tell us if you agree.

From left to right: ***Jim Besemer****, Project Manager, Specification & Design, User Interface;* ***Skip Russell****, System Primitives, Function Library, Error Recovery, Operating System Interface, Configuration and Setup;* ***Pat Thompson****, Extension Language/Compiler and Debugger;* ***Robynne Major-Jones****,* ***Howard Mercier*** *(not shown) Quality Assurance and Testing;* ***Eric Johnson****, Editor Core, Help Facility;* ***Milo Wadlin****, User Documentation.*

For two years this team worked in secret to create the world's best editor. We believe they succeeded magnificently. Now we'd like to know what you think.

New development environments and the size and complexity of projects make your 80's editor inefficient and obsolete. So we built you an editor for the 90's.

Try the Sage Professional Editor for thirty days with no risk. Pound on it. Compare it to any editor you've ever used. Then call our team at 1-800-547-4000 and tell them how you like it. There's no risk to you, no obligation. Not much risk for us either. We won't be able to pry it away from you.



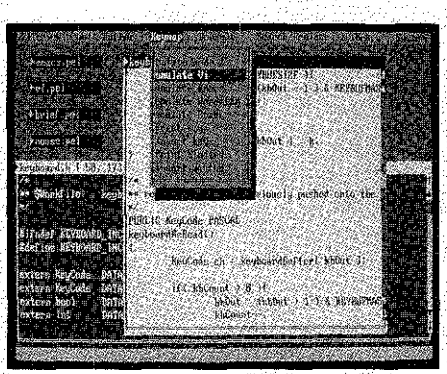
Works with or without a mouse.
Packaged with or without a Microsoft® Mouse.

Announcing the Sage Professional Editor

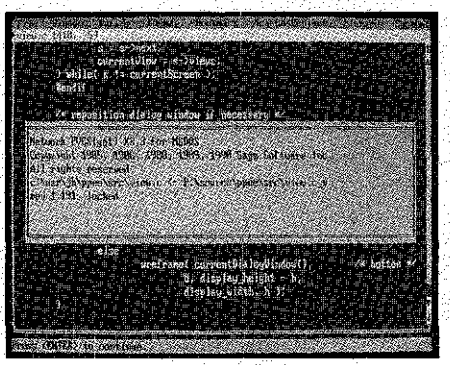
Right out of the box you'll be more productive with our editor than any you use today. The instant installation, elegant mouse support, advanced user interface, and point-and-shoot help get you running immediately. If you prefer the commands and keystrokes of a popular editor, our turnkey emulations duplicate them precisely, and you still gain the Sage Professional Editor's advanced features, windowing capabilities and powerful engine.

Later you'll make this editor truly yours by configuring the interface as you prefer. Every feature can be turned on or off as you like - from a clean screen, to tiled windows, to overlapping windows in various colors, pull-down menus, rulers, scroll bars, and line numbers - choose any or all and place them as you like.

Use the editor with or without a mouse - all functions are available without lifting your fingers from the keyboard. But the click-select windows, scroll bars, zoom, shrink-to-icon, block text selects, and other speedy mouse actions make editing extremely efficient.



Make our interface what you prefer, from clean screen to multi-window with drop-down menus and icons.



The editor environment provides seamless integration to the Polytron Version Control System (PVCS) or any other tool you care to connect.

Sage Software, Inc., 1700 NW 167th Place, Beaverton, OR 97006, (503) 645-1150, Fax (503) 645-4576, Int'l. (301) 230-3307, Int'l Fax (301) 984-3047.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pop open the DOS window and the editor shrinks to just 4k. So you can back-task to compilers and other tools without leaving the editor.

This package is stuffed with value. It includes MS-DOS, OS/2 and Dual Mode versions on both 5 1/4" and 3 1/2" disks, templates for popular languages, and you can buy it with or without a bundled Microsoft® Mouse.

But the reason you will become addicted to the Sage Professional Editor is the power behind that pretty face. We looked under the hood of other editors and found wimpy editing engines dressed up with all sorts of bells and whistles. Unacceptable. We wanted an industrial strength engine. So we started with a powerhouse virtual memory system that allows us to edit huge files (up to 100MEG) in as many as 256 windows - over two billion lines. It makes maximum use of available memory (including EMS) and uses advanced processes to safely minimize file I/O. All higher level services use this powerful VM scheme. Consequently, there are no size constraints on the macro library and no limit to Undo/Redo. You can have 1000 bookmarks, anchors and saved positions per buffer.

TurnKey Emulation Of

Vi | Brief | Wordstar | Epsilon/EMACS | Others

Sage Software, Inc. has acquired Polytron, the company that brought you PVCS and PolyMake. Now this group proudly introduces their Professional Editor under the Sage banner.

DEVELOPMENT POLYTRON TOOLS

"When I saw how well my mouse works with this new editor I had to try it. I became a convert when I saw the power of the extension language. This editor can be whatever I need it to be. I couldn't give it up."

Terry Hamm,
ImageBuilder Software

And then there's the extension language. The Sage Professional Editor uses a C-like extension language (100 percent PolyAWK compatible) and compiler/debugger that programmers find immediately intuitive. The function library gives access to nearly everything. From mouse click to menu, from word processing to external hooks. You can build the environment you need or want with the editor as the front end to your favorite tools. The seamless integration of the Polytron Version Control System (PVCS) is a sterling example of how cleanly you can hook external programs.

Emulations of Vi, Brief, EMACS, and WordStar, were written with the extension language. The source code for each is included. If you are addicted to some obscure editor you can modify a copy of emulator source and quickly make your own. Source for the menuing system, mouse interface, help and many other services is also included, so you can deeply modify the editor to be exactly what you want. Brief users who have a library of macros can use our translator to recompile them into our more useful and powerful C-like extension language.

It's easy to make the Sage Professional Editor a standard for your team or even the entire company:

1. Turn-key emulation means every programmer can use it immediately. Macros written to aid users apply to all emulations.

2. Provides a framework for integrating all your development tools into a single, consistent operating environment.

3. MS-DOS, OS/2 & Dual Mode versions are all included in the same box so *your* team can move into the 90's with the same editor.

4. Licenses are available for you, your team, division or company. The initial cost is low, and you continue to save as you add users.

MS-DOS | OS/2 | Dual Mode — All in One Package

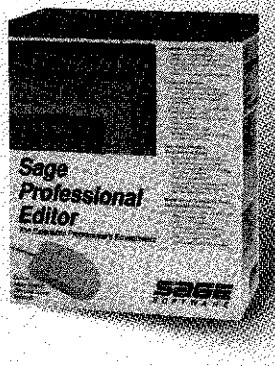
$295 U. S. List Prices. Single User Version.

$395 Includes a Microsoft® Mouse.

Specify serial, bus or PS/2 mouse. A 5-User License (without mice) is only $1,032.50. Call for prices on larger licenses. **Visa, MC** & P.O. orders:

1-800-547-4000

30 Day Money Back Guarantee



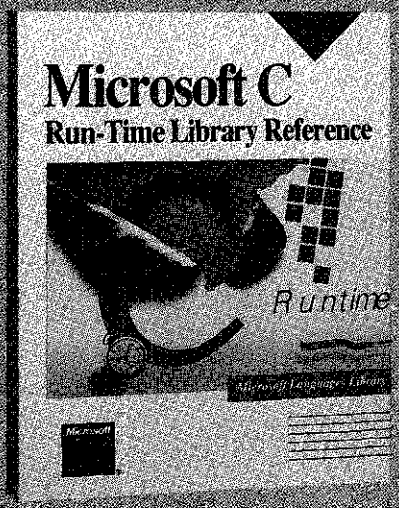
SAGE SOFTWARE

Circle 26 on Reader Service Card

PRODUCED BY BBS-PCBOARD.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Covers version 6

The Essential Reference to the Industry-Standard C Library



This superb reference to the Microsoft C version 6 runtime library (and the QuickC 2.5 runtime library subset) is both an overview of the rules and procedures for using the library *and* a reference to every runtime library function. The more than 500 functions address everything from presentation graphics and low-level system calls for the DOS programmer to multiple-thread support for the OS/2 programmer. The information in this reference shows you how to use the extraordinary power of these functions to simplify scores of common programming tasks, save hours of development time, and gain ready access to operating system functions. Of special note is the information on the expanded graphics library in version 6 and on MS-DOS,® ANSI C, OS/2, UNIX,® and XENIX® compatibility. $21.95

Microsoft Press
The Authorized Editions

Available wherever books and software are sold.
Or place your credit card order by calling 1-800-MSPRESS.
Refer to ad #AU5.

FORTRAN. Lahey F77LEM/32 generated code that was only about 50% slower. The difference in file I/O from the fastest 386 compiler to the slowest varied by about 300%. The difference in number crunching execution was at worst 50%.

For each FORTRAN compiler, I used any available switches or options that supported optimization. For your own Salford compiler, I always forced integer and logical variables to 4 bytes.

Finally, thank you for setting the record straight on the DBOS acronym. Unix influence seems to get around.

X WINDOW FOR DOS?

I am writing in reference to your article "X Window—A View from DOS" from the July/August '88 issue of *Programmer's Journal*. I have since heard a lot about X Window and would like to know more.

Since I only have access to a PC and want to learn to program in the X Window environment, I am interested in what became of your attempt to port X Window to the PC. Is an X Window library available for the DOS world? Any information you can provide would be greatly appreciated.

Peter Kerr
Washington, DC

THE EDITOR RESPONDS: *In this issue of* PJ, *John Otken writes about using the X Window protocol for text windowing. I have been working on several versions of an X Window server for DOS. These versions are designed to provide an X server in real-mode that communicates with an X client operating under a 386 DOS extender in the protected mode. I'll describe this material in an upcoming* PJ *issue on graphics.*

A number of vendors offer X server implementations under DOS. These X servers are designed to allow a DOS machine to network into a Unix machine running the X client application. These vendors include Graphic Software Systems (Beaverton, OR), PC Tech (Lake City, MN), Integrated Inference Machines (Anaheim, CA), and Locus Computing. Quarterdeck has also announced DESQview/X, a future version of DESQview that will incorporate the X Window protocol to directly support graphics. PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Language

AI Expert Systems
ARITY Combination Pkg 974
Exsys Professional 694
PC Scheme LISP by TI 74
Turbo Prolog Toolbox 70
TURBO PROLOG V2.0 110
VP Expert 219

Assemblers
Advantage Disassembler 279
ASM Flow 89
MS Macro Asm 97
OPTASM 129
Re:Source 129
Turbo Assembler/Debugger 98
Visible Computer 80286 84

Basic & Addons
DB/Lib 120
MS Basic 7.0 339
MS QuickBASIC V4.5 69
Probas by Hammerly ..... call
ProScreen by Hammerly ..... call
QBase 124
QuickPak Professional 139
Softcode 69
True Basic 69
Turbo Basic 69

C LanguageCompilers
High C 386 875
Lattice CV6 189
Microsoft QuickC w/QASM 135
MSC 6.0 345
Top Speed C Std 179
Turbo Cby Borland 97
Turbo C Professional 165
Watcom C8.0 386 std. 719

Cobol
MS Cobol V3.0 629
Microfocus COBOL W/tools 1499
Realia COBOL 844

Fortran
For_C FORTRAN'77 to C 670
Lahey FORTRAN F77L 525
Lahey Personal FORTRAN 84
MS FORTRAN with Codeview 295
RM Fortran 494

Modula2
Logitech Dev. Sys 195
Topspeed 3 pak 159

Object Oriented Programming
ACTOR 559
Whitewater Resource Tool Kit 160
Smalltalk/V 286 169
Turbo C++ 109
Turbo C++ Prof 159
Turbo Debugger & Tools 99
Zortech C++ with source 205
Zortech Dev't Kit 380

Other Languages
IntegrAda 724
Janus Ada 369
Meridian Ada Dev't Kit 980
MuLISP87 219
PC/Forth+ 199
Personal Rexx 129

Turbo Pascal
Turbo ASYNCH PLUS 115
Turbo Pascal 5.0 105
Turbo Pascal 5.0 Professional 175
Turbo POWER TOOLS PLUS 124
Turbo Programmer 444

DBMS

Data Base Development
DBXL 199
Clarion Prof 525
Clipper S87 399
dBASE III Plus 450
Force 395
FoxBASE + ... 239
Clipper 5.0 495
Fox Pro Lan 745
Fox Pro SU 579
QuickSilver 379

dBASE Library
Artful 189
CodeBASE IV 215
Funky 175
IDL 125
Netlib 229
Prodlp V3.5 119

File Management
Btrieve 180
Cindex Plus 329
CISAM 325
CQL wSRC 349
WPASS 349
ctree ..... call
dBC III PLUS 435
db_FILE RETRIEVE 265
Essential BTree w/source 149
FABS Plus 172
Netware C interf. 239
Netware SQL 469
pBase 185
Turbo Programmer/C 449
WKS Lib 189
XDBC 555
XQL 599
Xtrieve PLUS 459
Btree Filer 97

Addons

C Communications
New! C Async Lib. sy Silverware ..209
BreakOut II 99
C ASYNCH MANAGER 155
Essential Communications 295
Greenleaf Comm Library 279
Greenleaf ViewComm 315
Lattice Comm. Library 179

Dbase Addons
AdComm 99 105
BLINK 179
CLEAR + FOR dBASE 139
Dbase Online V2.0 125
dBugsource debugger 170
dBXdBASE III to C 460
dClip 169
DCS Report Gen 265
desQ Clipper Api 469
dGEbusiness graphics 259
Documentor 225
dQuery 165
dSalvage 79
FLIPPER Graphics Library 175
Spellcode 89
Touch & Go 325
R&R Code Generator 125
R&R Relational Reportwriter 125
Scrimagescreen/menu 129
SilverComm Library V2 169
Silver Paint Library 79
Steve Straly's Tools 169
UI ProgrammerDev's Version 2.0 ..419

General Addons
C Tools PlusV5.0 105
C Utilitiesby Essential 154
Greenleaf Functions 159
Greenleaf SuperFunctions 209

Graphic Addons
Essential Graphics 345
Graphic C 329
GraphicsMenu 159
Greenleaf MakeformDOS 90
GSS Dev't Toolkit 505
Halo '88140 + dvices 275
Menuet 259
MetaWINDOW/PLUS 269
Mewel 149
PCX Programmer's Toolkit 175
Slate w/Graphics 415

Text Screen Addons
AEWindows 204
C Worthy w/forms 259
Cursesby Aspen Scientific 125
Greenleaf DataWindows 325
HiScreen XL 124
HAM by JYACC 464
VC Screenpainter 114
Vitamin Csource, menus 164

Utilities
BACKIT 120
COPY II PC 35
Disk Technician Advanced 149
FASTBACK Plus 142
MACE GOLD 129
MACE Utilities 90
Norton Utilities Advanced 101
PC/Tools Delxe 6.0 95
V opt 111
XENOCOPYPC 70
XTree Pro 111

Miscellaneous

Other Products
Aspen Korn Shell 109
CO/SESSION 214
DESQ 386 ..... call
dQuery/Lib 274
Line & Locate ++-ROM MSC. . 339
Printware 149
Smartern 240 294
Stingray Clarion Lems ..... call
QEMM 386 55
386 Maxpro 115

Case & Prototypes
Demo II 155
NEW! Easy Case + ... 205
Instant Replay/Nostradamus .. 134
Interactive EasyFlow 135
MetaDesign by Meta Software 329
ProC 425
PROTEUS Prototype System . 124
Show Partner F/Xdemos 315
Softdemo 59
Protoview 499

Tools

Debuggers
386 DEBUG 155
MultiScope 135
Periscope OK 465
Softprobe 86/PX 345

Development Tools
DASM 239
MKS AWK 84
MKS Lex & Yacc 205
MKS RCS 155
PC Lint ..... call
Plink 86 Plus overlays 399
PolyMake 129
PVCS Corporate 355
.RTLINK + by Pocket Soft 409
TLib 89

Editors
BRIEF ..... call
Epsilon 155
KEDIT 119
Me wsource 169
MKS VI 125
MultiEdit 90

★ SAGE PROFESSIONAL EDITOR ★
Mouse Support, DOS/OS/2,Emulates
Brief, vi & Emacs, EMS Support Fully
ProgrammableList 295 Ours 249
30day Money Back Guarantee

NROFF/PC 117
PC/EDIT+ 269
Qedit 79
SPF/PC 189

Windows & OS/2

Windows
Asymetrix Toolbook ..... call
Case:W 754
CTalk/views 375
Dbfast/Windows 325
DialogCoder 429
MS Windows SDK 325
MultiScope Windows 285
Object Graphics 319
ProtoView 619
RF Flow 69
WinTrieve 335
WNDX Gui Tools 445
Windows 3.0 99

OS/2
Brief ..... call
Btrieve 475
C Tree 309
C Worthy 339
Greenleaf Data Windows 319
Mks Awk 159
Mks Lex & Yacc 339
Mks Toolkit 399
Multiscope 340
OS/2 PM Toolkit 359
VC Screen 139
Vitamin C 289
PC Yacc 620
Zortech OS/2 129

1-800-323-1809

PROGRAMMERS WAREHOUSE

7819 East Greenway Suite 8 • Scottsdale, AZ 85260 • Outside US call (602) 443-0580

All Prices and Offers Subject to Change Without Notice

Order Now! Shipping Same Day!

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software Zealots

Is software your life? Then you'll find life doesn't get any better than Microsoft. We were founded on a passionate belief in the role software would play in personal computing. We need similarly fixated individuals to help create our next generation of revolutionary products.

You'll work in small, fast-paced groups with people who share your fervor, under a management legendary for its accomplishments in microcomputer systems, applications and languages. And you'll enjoy everything you need to create your own vision, including a door, attached to an office, for those times of quiet contemplation.

Technical Leads

Your design expertise will drive a team of experienced developers, shaping future in-house tools, sophisticated help systems, databases and windowing applications. We're also developing projects in languages and handwriting recognition — and some truly visionary ideas we can't reveal yet. Qualifications include a successful record of software development and technical lead experience on PC products.

Microsoft is out to change the world, but not the part we live in: Redmond, Washington. Minutes away from Seattle, we're surrounded by two National Parks, the San Juan Islands, scenic mountains, in-city lakes, and year-round greenery. All this and the clean air to see it.

Show a little passion.

Send your resume and a cover letter that reflects your desire to shape the world of personal computing to: Microsoft Corporation, Recruiting, Dept. WCSZ-0990PJ, One Microsoft Way, Redmond, WA 98052-6399. No phone calls, please. We are an equal opportunity employer and are working toward a more culturally diverse workplace.

Microsoft®
Making it all make sense™

Circle 232 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Cache Memories

By Hal W. Hardenbergh

Caches are small, fast memory systems used to make DRAM memory look faster than it really is. This article will cover basic information and terminology about caches and ponder their effect on computer systems.

We need caches because a personal computer with a 50MHz CPU needs a new instruction every 20 nsec. DRAM cycle time (which is slower than access time) is typically 190 nsec, a mismatch of about one order of magnitude.

Caches work because of locality of program instruction references. Any current program instruction is probably part of a loop, subroutine, or function call, and will probably be used again soon. That's locality in time. Those instructions that have memory addresses near the current instruction will probably be used soon, too. That's locality in space.

Caches don't work for *every* instruction fetch. However, if the cache works most of the time (and 96%+ rates are possible with a 64K cache), it will provide a significant speed advantage. What do we mean by "working"? If we find a copy of the next instruction we need in the cache, then it works for that instruction (a "hit"). If we don't, then it didn't work (a "miss").

We want a high hit rate and a low miss rate. That is, finding the next instruction in the cache most of the time. And although caches have a higher hit rate for instructions than for data, they do work for both.

Caches work with physical or virtual addresses. The three types of caches that use physical addresses are classified as either direct-mapped, fully associative, or set-associative.

DIRECT-MAPPED CACHES

"Direct mapping" means that a given location in the main RAM memory can be cached in only one location in the cache system. That location is determined by the "index," the low order address bits used to address the cache tag RAM. (More will be said about cache tags in a moment.)

ILLUSTRATION BY JIM CARPENTER

Adding a direct-mapped cache to a processor means adding three subsystems: a cache data RAM, a cache tag RAM (which includes a comparator), and the cache controller (which might be a programmable logic device).

Figure 1 shows a processor with a very simple direct-mapped cache system. Let's assume that the CPU is a fast 8-bit 6502, circa 1975. The main memory is 32K, mapped into the lower half of the 6502's memory space. You may remember that the way to slow a 6502 down was to stretch the clock rather than adding wait states, so we've included a clock stretcher.

The RAM memory sits in the lower half of the 64K address space, with the bootstrap ROM and I/O in the upper half. Our cache controller looks at the most significant address bit, A15, so that it can ignore any memory cycle which doesn't address the RAM in the lower half of memory.

The cache data RAM holds 256 8-bit words and is addressed by A7-A0. When a hit occurs, the cache controller allows the output port of the cache data RAM onto the processor's data bus. When a miss occurs, it must wait for the slower main memory.

In Figure 1's example, the cache tag RAM also holds 256 8-bit words. One bit is the "valid entry" bit, and the other 7 hold the "tag" for the corresponding entry in the cache RAM (in this case A14-A8). The cache tag RAM has a comparator to compare the tag to A14-A8 for the current index. The current index (A7-A0) is used to address the cache RAM (and, in this example, the cache data RAM). *If* the indexed tag has the valid bit set and *if* the tag matches A14-A8, then we have a cache hit.

In the event of a cache miss during an instruction fetch, the cache controller reads the data from the main RAM and writes it into the indexed cache. It also writes the address tag bits and the valid bit into the cache tag RAM. (The most significant bit of the tag RAM is tied to +5V so that any write into the cache tag RAM sets the valid data bit.)

CACHE TAGS

Cache tags tell whether a referenced location in main memory has been cached. At a minimum, a tag must contain a bit to indicate a valid cache entry (at system power up, there are no valid entries). It must also contain a portion of the address of the cached entry, if any.

In a direct-mapped cache, the size of the portion of the address which must be a part of the tag can be easily determined by dividing the number of cacheable locations in the main memory by the number of words in the cache tag RAM. In Figure 1's example, we divide 32K (or 32,768) by 256, giving us 128. So we need 7 address bits in our cache tag, that being the number of bits needed to decode 128 locations.

The tag field can contain other specialized identifier bits as well, depending on the design of the system. One such identifier will be discussed later.

FULLY ASSOCIATIVE CACHES

"Fully associative" means that *any* location in the main RAM can be cached *anywhere* in the cache data RAM. Note

Hal W. Hardenbergh is a senior hardware engineer who sometimes programs. Hal works at International Imaging Systems in Milpitas, California.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

that this is the exact opposite of direct mapping. When an entry is written into a fully associative cache, the oldest entry is the one that gets replaced.

This flexibility carries a high price. Each cache tag has a valid bit plus the entire address of the cached entry—there is no index. Worse, a comparator is needed for each cache entry. For these reasons, fully associative caches are not used. However, an understanding of fully associative caches serves as a useful introduction to set-associative caches, which *are* commonly used.

SET-ASSOCIATIVE CACHES

While a fully-associative cache allows any location in the main memory to be associated with *any* location in the cache, a set-associative cache permits any location in the main memory to be associated with *only one* location in *each set* of the cache. Set-associative caches use N cache tag RAMs. N is generally 2, 4, or 8, with the lower numbers more common. If the cache is 4-way set-associative, then each location in the main memory can be associated with four locations, one in each of the 4 cache tag RAMs.

Each cache tag RAM is used as if it were part of a direct-mapped cache. What's different is that now each location in memory has N possible storage locations in the cache data RAM. The 486 and the 68040 both use 4-way set-associative internal caches.

Since each cache tag RAM contains a comparator, an N-way set-associative cache has N comparators. The cache controller has to monitor the outputs of all the comparators to determine whether a hit has occurred.

To make matters worse, the cache controller has to decide where to store the data from a miss—that is, in which set? For this task, extra circuitry is needed. Most folks agree that the least-recently-used (LRU) algorithm is best suited to make this decision, but that requires some extra RAM to store the usage history. While a rotating priority is much simpler, it still requires some storage.

If the value of N in an N-way set-associative cache is 1, then we have a direct-mapped cache. If N is equal to the total number of cache tags, then we have a fully-associative cache. There is little difference between these two cache types, provided we stay with physical addressing and away from virtual addressing.

CACHE LINES

In Figure 1's simple example, there was one cache data word—an 8-bit word—per cache tag entry. Normally, there will be more than one. The cache designer gets to choose this number.

In practice, most programmers use four cache data words per tag entry, with the word width determined by the processor's data bus width. There are very few exceptions to this rule. The four data words per tag entry are called a cache "line" (sometimes this is called a "block").

Modern DRAMs feature either nibble, page, or static column mode. Using the burst read mode available on these DRAMs, it takes less time on a cache miss to read in a 4-word line once than it does to stop 4 times on 4 misses and read in one word each time. Also, the cache tag RAM size is reduced by a factor of 4.

Because there's no significant improvement in speed for an 8-word DRAM burst over a 4-word burst, 8-word cache lines are not used much. Indeed, today's nibble mode DRAMs are only capable of 4-word bursts, maximum. Bursts longer than 4 words would lock out other processors or DMA devices from memory for longer periods of time. And there is a greater likelihood that parts of the 8-word line would never be used. Both the 486 and the 68040 use 4-word, 16-byte cache lines.

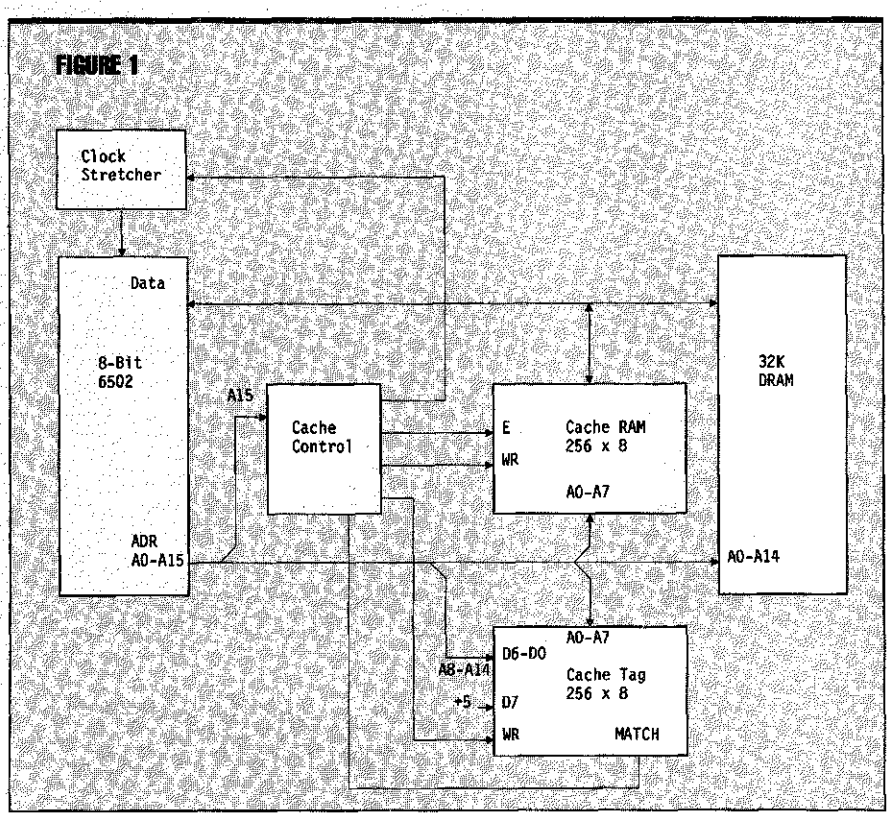
UNIFIED OR HARVARD?

Cache systems can be either unified, in which instructions and data are contained in the same cache, or Harvard, with the instructions and data contained in separate caches. In a Harvard system, the processor sees the instruction cache as a read-only device. Data caches and unified caches are read-write devices.

Intel x86 processors and IBM 370s must use unified caches. There's no choice. The reason is the same for both: so much software out there uses self-modifying code, including Microsoft Windows.

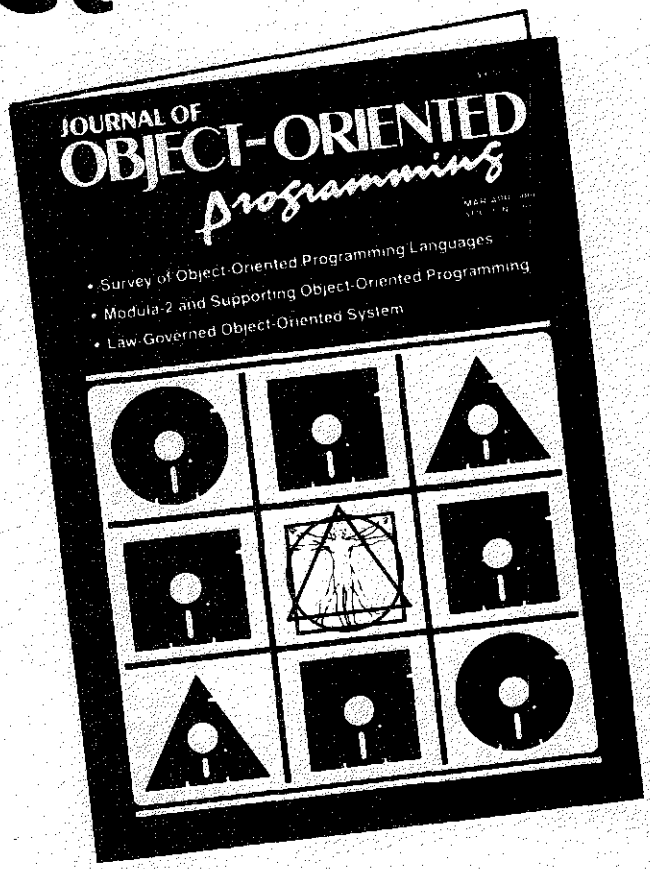
Harvard processors—the 68040 and all of the RISC processors—have an architectural advantage over the Intel x86

FIGURE 1



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Object is the Subject



The Journal of Object-Oriented Programming

The only publication in the world devoted exclusively to object-oriented programming

featuring

C++, Smalltalk, Objective-C, Eiffel, Actor, CLOS, Object Pascal, and other new OOP languages

In every issue you get:

- peer-reviewed articles
- groundbreaking research
- coding tips & algorithms
- "insider" info
- training services listing
- actual program listings
- latest product news
- candid product reviews
- thought-provoking columns
- career opportunities

Here's a sampling of upcoming in-depth articles:

- "A Survey of Object-Oriented Languages"
- "Constructing Abstractions for Object-Oriented Applications"
- Variables Limit Reusability
- Object-Oriented Languages Extensions for CAD/CAM
- "Object-Oriented Software Reuse: The Yo Yo Problem"
- Sharing and Organization Protocols in Object-Oriented Programming
- An Object-Oriented Programming Environment For Advanced Database Applications
- "A General Model For Concurrent and Distributed Object-Oriented Programming."
- "Adequate Testing and Object-Oriented Programming."
- "Class Diagrams: A Tool For Design, Documentation and Teaching."
- "Automatic Differentiation on C++."
- "Security in Object Oriented Data Systems."

Authoritative, Technical, Code-Intensive

Get the latest developments in this new programming methodology.

Order by Phone (212)274-0640
by Fax (212)274-0646

JOOP **Enter my subscription at the term below.**
1 year — 6 issues

❑ Individual $49 ❑ Institution $119

Outside the U.S., add $33/yr air service

❑ Check enclosed (payable to JOOP) ❑ Bill me

Foreign orders must be paid in U.S. dollars drawn on a U.S. bank

Charge my ❑ Visa ❑ MasterCard

Card # _____ Exp. _____

Signature _____

Name _____

Company _____

Address _____

City _____

State _____ Zip _____

Phone (Day) _____

Mail to:
JOOP, Subscriber Services, P.O. Box 3000, Dept OOP, Denville, NJ 07834

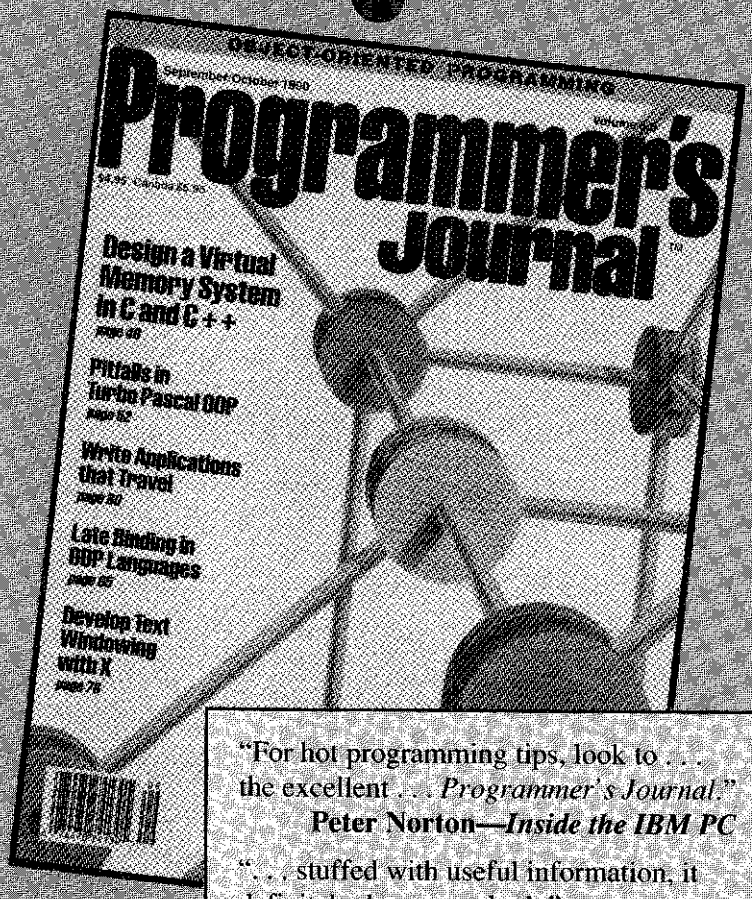
DOJO5

Circle **205** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"...love this magazine!"*

Industry experts agree that when it comes to a truly useful programming magazine, ***Programmer's Journal*** just can't be beat. Recognized by the **Computer Press Association** in 1989 for excellence in journalism, ***PJ*** is "... full of practical information and guidance presented with superb clarity and perspective." Try ***PJ*** and judge for yourself. It's easy—**just call toll free.** Start receiving 6 issues a year for only $19.95.


OBJECT-ORIENTED PROGRAMMING
Programmer's Journal
Design a Virtual Memory System in C and C++
Pitfalls in Turbo Pascal OOP
Write Applications that Travel
Late Binding in OOP Languages
Develop Text Windowing with X

"For hot programming tips, look to ... the excellent ... *Programmer's Journal.*"
Peter Norton—*Inside the IBM PC*

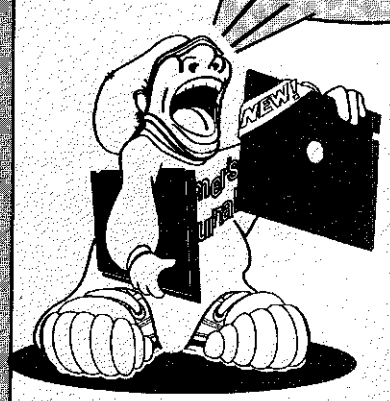
"... stuffed with useful information, it definitely deserves a look."
Ray Duncan—*Dr. Dobb's Journal*

*"I really love this magazine!"
Paul Mace—Paul Mace Software

To subscribe call toll free:

800-234-0386

PROGRAMMER'S JOURNAL
DiskScription
Six *PJ* Code Disks now only—
$39.95
1 Year (6 issues)



All Our Code— On Disk

Typing in long code listings uses up a lot of valuable time. And calling up bulletin boards is not exactly a free ride. So what does the modern programmer do? Get a ***PJ* Code DiskScription,** of course!

Each disk contains all of the program listings published in ***PJ*** plus .EXE files, .OBJ files, benchmark source files and more! If purchased separately the regular price for a year's worth of ***PJ*** code (6 disks) would be $60.00. **You can save yourself some time, effort and over $20.00 by ordering your *PJ* Code DiskScription today!**

*Guarantee: If you are dissatisfied with your **PJ Code DiskScription** for any reason, we will gladly send you a full refund on all unmailed diskettes.*

All *PJ* Code DiskScriptions must be prepaid. Return the card on the opposite page with payment. Or, for faster service with your VISA MC call toll-free: 800-234-0386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and IBM 370s because they can fetch an instruction *and* a data word in the same cycle. Intel and IBM can equalize this advantage any time they want by turning their backs on their respective software bases, each of which is 87 miles high. In fact, Intel did just that with the i860. Uh, what applications do you run on *your* i860?

(To my considerable embarrassment, I wrote a sidebar for an article in *PJ* which predicted that the Intel 586 would have Harvard caches. I know, and knew, better. I'd feel really bad if it weren't for that article in *The Atlantic* magazine a year or so back pointing out that most articles contain at least one error.)

WRITE-THROUGH AND -BACK

Most processor memory cycles are instruction reads. Data reads, though less common, are treated just like instruction reads. Because caches work so well on instruction and data reads, it's ironic that most of the main memory bus traffic in a cached CPU system consists of data writes—a can of worms.

There are two basic ways in which data writes in cache systems can be handled. One is write-through, in which the data is written both into the cache RAM and also into the main, slow, RAM. In the other, write-back, the data is written only into the cache RAM. Write-back then writes into the main RAM later, when the cache line which contains the data is invalidated (overwritten by a new cache line). There are disadvantages to each approach. The more complicated write-back is primarily used to reduce bus traffic in multiprocessor systems.

Some companies, and some articles, use other terms for write-through and write-back (write-back is often called copy-back). The question is, is the data written immediately into main memory or not? If another device can access the main memory, this distinction could be important.

DIRTY CACHE LINES

A cache using write-back needs an additional bit in the cache tag field to indicate whether the cache line contains data. (It's data if the line has been written from the CPU.) If the bit is set, meaning that data has been written into the cache line from the CPU but not yet written back to memory, the cache line is said to be "dirty." Before an existing cache line can be invalidated, the line must be written back to memory so that a new cache line can be written into the cache RAM.

As you can see, cache controllers that support write-back are more complicated than those that merely support write-through. The 486 just supports the write-through protocol, which makes sense for a CPU intended for the personal computer market. The 68040, which Motorola seems to be aiming at the workstation market, uses the more complex write-back protocol.

POSTED WRITES

When the 486 does a write, it writes into one of four write buffers and immediately proceeds with the next instruction without waiting for the data to be written to DRAM. Cache fills due to a cache miss are given priority over data writes from the write buffers. This increasingly

PERISCOPE® IV
THE Real REAL-TIME SOURCE-LEVEL DEBUGGER
New Version 5

With Periscope Model IV, you can debug your software while it runs at full speed, something *no* software-only debugger can do.

You can also examine the execution history of a hardware interrupt in Periscope IV's real-time trace buffer, something *else* no software-only debugger can do.

When The Going Gets Tough... Periscope IV helps you debug most any program, but is specially designed to debug programs that software-only debuggers can't. Model IV users use it to debug TSRs, ISRs, device-drivers, DOS, BIOS, communications software, real-time data acquisition programs, multitasking software, network software, keyboard drivers, disk caching software, systems software, spread sheets, EMM products, and so forth.

"Periscope is my #1 favorite program... I would rather change my editor than my debugger," writes Phil Mayes, who used Periscope IV to track down some very elusive bugs causing intermittent corruption.

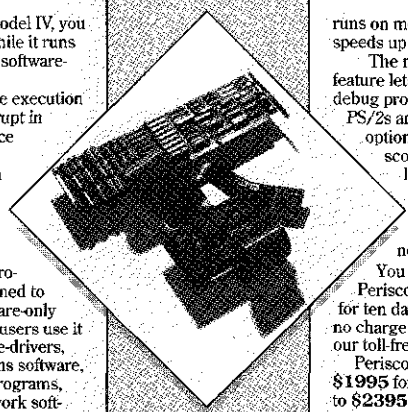
Periscope IV provides source-level support for popular PC compilers and linkers, such as those produced by Microsoft, Borland, and others, and it runs on most any 80286 or 80386 at speeds up to 25MHz.

The new Remote Debugging feature lets you use Periscope IV to debug programs running on IBM PS/2s and compatibles. The optional Plus board keeps Periscope from using any of the lower 640K, so Periscope can't be overwritten and doesn't use the memory your program needs.

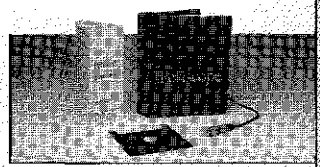
You can try Periscope IV for ten days at no charge before you buy it. Call our toll-free number for details. **FREE 10-day Evaluation**

Periscope IV prices range from $1995 for a 16 MHz 80286 system to $2395 for a 25MHz 80386 system. The optional 512K Periscope PLUS board is $400.

Given the value of your time, can you afford *not* to try it?



The Periscope manual, software, breakout switch and quick-reference card are included with Model IV.



"The Model IV hardware really makes Periscope shine, especially when you've got timing-related problems. I can now track down changing pointers and altered buffers on my 386. I've been using it to debug Crosstalk Mk. 4 and there's just no better way to do it."

JEFF GARBERS
Director of Software Development
Crosstalk Communications

Call Toll-Free Today for More Information or to Order Your Periscope: 800-722-7006

In the UK call Roundhill Computer Systems at 0672 84 535. In Germany call H+B EDV at 07542 6353 or ComFood at 02534 7093. In Sweden call LinSoft at 013 124780.

The Periscope Company, Inc.
1197 Peachtree St., Plaza Level
Atlanta, GA 30361
404/875-8080 • FAX 404/872-1973

Circle 6 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

common technique to reduce bus congestion is called a "posted write."

Alas, nothing is as simple as it seems: memory bus reads and writes can be "out of order" as compared to the 386, and that in turn means that the self-modifying code in Windows might not run on the 486. So the 486 CPU contains some additional circuitry that performs checks to find out if a read is being performed to a location which has been posted but has not yet been written back to memory (this is a limited form of "bus snooping").

ABOUT BUS SNOOPING

If a DMA device or another processor has access to main memory, then using write-back can be a real problem because the data value that the cached processor thinks is in memory really isn't; it's still sitting in the cache. When the DMA device or the other processor looks for that data value in memory, the correct value is not there. Bus snooping is a way of fixing this problem.

Here's how bus snooping works: the cache system contains additional circuitry to monitor activity on the system address and data bus. If the additional circuitry detects a read on the bus, it checks to see if that location has been cached and is dirty. If so, the additional circuitry halts the system bus and writes the cache line back to main memory (hence, "write-back"). The other device recognizes the bus halt and knows it must retry the bus cycle. If the cache line contains instructions, there's no problem.

If the additional circuitry detects a write into system memory, it checks to see if that location has been cached. If so, the cache tag has its valid bit cleared, whether the entry is a data word or an instruction.

Bus snooping was first used six years ago in the Synapse computer, which used up to 32 cached 68000s. Five years later, the 486 became the first single-chip microprocessor to support bus snooping.

SNOOPERS BEWARE

If a cache system has a 2.5% miss rate, and if the main memory is 8 times slower than the cache, then the CPU will utilize 20% of the main memory bus bandwidth (this is simplistic, but let it go for now). That means that a multiprocessor system with bus snooping can put a total of 4 CPUs on the main memory without strangling the main memory bus, right? And now we have a system which is 4 times faster than a single CPU, right?

Well, I guess so. I've seen a multiprocessor system based on 4 Motorola 88000 RISC CPUs, each of which has a claimed performance of 17 MIPS. The performance claim of the multiprocessor system was 68 MIPS.

Hmm. There is this small problem: a RISC chip needs to use the cache tag RAM on just about every clock cycle to feed the next instruction to the CPU, and the bus snooper has to use the cache tag RAM on every memory bus cycle. Obviously they cannot both use the cache tag on the same clock cycle.

The simple fix for this is to provide a duplicate cache tag which will be used exclusively by the snooper. There is also

Instant-C

INCREMENTAL C COMPILER

"An Utterly Addictive Speed Improvement"

B. Rubin, *The C Gazette*, Autumn 1989

Speed Microsoft™ C development. *Instant-C* is a Microsoft-compatible incremental compiler. It can rebuild 100,000 line programs in seconds. And you can still use Microsoft C's optimization.

Find bugs other compilers miss. *Instant-C* debugs as fast as it compiles. Here are some bugs that (in a Microsoft C program) could hang your computer. *Instant-C* catches them automatically:

- Array reference out of bounds
- Use of uninitialized pointer
- Read/write through null pointer
- Stack overflow
- Clobbering the program or DOS

For those few bugs that still elude detection, *Instant-C* integrates the debugger with the compiler for true source level debugging.

No slowdown on big programs. *Instant-C* uses our DOS/16M technology, so you can work on programs bigger than 640K with no swapping to disk.

Everything you need. *Instant-C* has much more to offer, like dynamic cross referencing and lint-like static checking. Support for third party libraries. Interactive testing of single functions or partial programs. Everything you need to program and debug faster.

Call 508-653-6006 for *free* information.

Rational Systems, Inc. 220 N. Main St., Natick, MA 01760 PJH1

Instant-C requires an 80286/386/486 and at least 1M of memory. Microsoft is a registered trademark of Microsoft Corp.

Circle 20 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

First impressions last.

INSTALL 3.0 gives your software product a professional introduction.

Installation procedures that rely on batch files and DOS commands not only look amateurish but also have limited error handling capabilities. Today's users expect quality and ease of use from software beginning the moment they open the package. A smooth, professional installation makes an important first impression.

Everything you need

INSTALL 3.0 is customized for your product with an ASCII text file called a "script file." INSTALL 3.0 comes with many sample script files you can modify and use immediately. No programming is necessary to create most installations, however C source code is included for your convenience. And technical support is free.

INSTALL 3.0

- Features very sophisticated error-handling for such items as bad disks, open drive doors, etc.
- Does high performance data compression for files up to 4.3 gigabytes
- Can safely and reliably modify user's CONFIG.SYS and AUTOEXEC.BAT files
- Detects all monitors, CPUs, DOS versions, numeric coprocessors, etc., to install a machine-specific product without asking the user questions
- Allows selective installation of parts of your program
- Includes source code
- No royalties

Fast and Simple

Use INSTALL 3.0 to create an elegant installation procedure that will increase users' confidence in your product. INSTALL 3.0 takes advantage of available RAM for lightning-fast file transfers. All your documentation has to tell the users is TYPE "`A:INSTALL`".

Proven Reliability

INSTALL 3.0 has been used for over four years by some of the biggest (and smallest!) names in the industry, in the U.S. and abroad, to install millions of copies of programs. Add your name to that list today.

$249.95 plus shipping
30 day money back guarantee
Free technical support • No royalties
MasterCard/VISA/COD/POs welcome

KNOWLEDGE DYNAMICS CORPORATION
Highway Contract 4, Box 185-H
Canyon Lake, Texas 78133

Call or FAX by 2pm and receive INSTALL 3.0 tomorrow!

Sales	1/800-331-2783
International	1/512-964-3994
24 hour FAX	1/512-964-3958
24 hour BBS	1/512-964-3929

Circle 295 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

a simple reason why this is not done—it increases the cost of those 99.99% of CPUs which are not used in multitasking systems. You will not see duplicate cache tags on a mass market microprocessor any time soon, if ever. (A recent *IEEE Micro* article which was written by Motorola about the 68040 specifically apologizes for the absence of a duplicate cache tag, explaining that it would be too costly.) Therefore, you should be wary of multiprocessor performance claims, even if and when anyone ever figures out how to program them effectively.

CACHE SIZE

The hit ratios that can be obtained by a given cache size and degree of set-associativity are well known. Table 1 lists a few values from reference 2. In articles written as recently as 1987, you will still see a 32K cache described as large. Today, we would call it small. Note that cache performance does *not* increase linearly with size. The time lost by the controller when it has to stop to service a cache miss is very important—and is different in various implementations. Remember that caches provide a statistical performance increase. The hit rates listed in Table 1 are averages taken over a large number of programs and computers. Keep in mind that your mileage *will* vary.

Tempus doth fugit mightily; reference 2 carries a 1990 copyright date, but does not list hit [miss] rates for caches larger than 128K—and today you can buy a 486 machine with a 256K RAM cache.

Many processors now contain significant on-chip caches. The 486 and 68040 each have 8K and the i860 has 12K. If an external cache is used, we have a two-level cache. The on-board cache is the primary cache, and the external cache is the secondary. Ironically, secondary caches are already common in 486-based personal computers, but are found in only the most expensive minicomputers.

FASTER CACHES?

In a direct-mapped cache, the cache controller recognizes a read into the main RAM memory space and immediately enables the output of the cache RAM on pure speculation. It will not know for a while whether the read cycle is a hit or miss, but most cycles are hits. If a hit occurs, the data from the cache RAM will arrive at the CPU more quickly because the cache RAM is already enabled.

Some microprocessors include hardware support for direct-mapped caches. The Cypress SPARC, for instance, issues an address on one clock edge and then latches the data bus on the next clock edge. The external cache controller has yet another half-clock to let the CPU know whether the data was valid.

For a given semiconductor technology, an external direct-mapped cache is about 30% faster than an N-way set-associative cache. However, this is most applicable to cache systems constructed of discrete ICs, where inter-chip delays dominate.

The internal caches in the 486 and the 68040 use 4-way set-associativity to maximize the hit rate in their 8K caches, yet they run at the full clock rate. If a set-associative cache is able to run fast enough, it's the preferred type. Isn't it just great to have a 1.2 million transistor design budget?

INTEGRATED CONTROLLERS

The older Intel 82385 is a semi-integrated cache controller ("semi" because external static RAM is required) for 386-based systems. It uses 2-way set associativity and only a 4-byte (one 32-bit word) cache line. The 82485 is semi-integrated as well, and is intended to support the 486. Intel is offering it as a stand-alone module, packaged with cache RAM.

Because Intel noticed that many 386 systems were being shipped with cache systems included, they have recently introduced their own fully-integrated cache chip for the 386. The 82395 chip contains over one million transistors, is nearly half an inch on a side, and comes in a 196-lead (!) package. It uses 4-way set associativity, a conventional 16-byte cache line, and includes 16K cache RAM. This is the first megachip ever introduced to support a microprocessor which is already obsolescent.

Both of the newer cache controllers support burst cache line fills (rapid 4-word DRAM accesses) and streaming. Streaming means that the first word of the cache line to be fetched is the one which incurs the cache miss, so processor execution can resume after that first word fetch without waiting for the rest of the cache line to be loaded. This is very important on the 486, where any wait state is an instruction that was not executed.

It takes a lot of chip real estate to support streaming. The 68030 and 88000 don't have large silicon budgets by today's standards and their performance is significantly degraded by the lack of streaming. The 486, 68040, 82395, and 82485 all support streaming, and all but the latter have more than one million transistors. (The 82485 has a mere half-million transistors, and is nothing but a semi-integrated cache controller!)

Caches made from discrete ICs tend to be quite large; however, they won't support streaming and sometimes won't support burst cache line fills. The new integrated cache chips (of which the 82395 is the first) have smaller caches (e.g., 16K) but support efficient transfers and replacement algorithms. Both types work very well.

Other companies are also offering cache support for the 386 and 486 chips. By all means check them out; there are no secrets to cache design. But producing BIG fast chips is another matter entirely.

CACHE BLOCKING

Any computer system with a cache needs a way to "turn off" the cache effect. For instance, I/O ports are always at the same address, but the CPU can't just cache the data in that location once and use it over and over. The latest microprocessors have the ability to enable or disable caching on a virtual-

TABLE 1: Hit Rate in % vs. Associativity and Cache Size

	1K	8K	32K	64K	128K
1-way	80.9	91.3	95.0	96.1	97.4
2-way	83.9	93.1	95.9	97.0	98.0
4-way	84.8	93.5	96.2	97.2	98.4

Source: see Reference 2.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

memory, page-by-page basis (that is, the 486 and 860 do—the 386 doesn't). Those which incorporate write-back usually have the ability to switch to the simpler write-through protocol (though the 860 is an exception). And microprocessors with internal caches will commonly have an external "cache inhibit" pin so that the hardware can inhibit caching at will (for instance, at certain memory locations).

VIRTUAL MEMORY

Virtual memory and its necessary memory management unit (MMU) add considerable complexity to cache design. For instance, the virtual-to-physical address translation can occur either before or after the cache.

Both the 486 and the 68040 provide support for virtual memory, but the translation is done prior to accessing the caches, so they use a physical address. The i860 internal caches are virtually addressed, which makes it a pig on context switches. The advantage is that the 860 has to perform the virtual-to-physical address translation only on cache misses. (UNIX has lots of context switches. Some folks haven't yet figured out whether the 860 would make a good UNIX engine!)

Virtual memory is worthy of further discussion, but not in an introductory article on caches. Reference 5 is a 16-page discussion of virtual memory and its implementation in modern microprocessors, including the Intel family. Let me just say that the nicest thing about virtual addressing using microprocessors is that you can always turn it off.

DO YOU NEED A CACHE?

If you have a fast 386 machine (25 to 33 MHz), then the answer is yes. If you're buying a 486 machine, it automatically has an 8K cache. Intel claims that the 486 has a hit rate of about 96% running MS-DOS and nearly 92% running UNIX or OS/2. Intel, which wants to sell secondary cache modules, claims that its module offers "up to" 15% more performance. Others have reported that an external cache on a 25MHz 486 provides a 5% speedup. When the 486 PC marketplace matures, you'll have to decide whether it's worth it to you to put out another couple hundred of your hard-earned bucks for a performance boost that small.

FUTURE PC TRENDS

To predict the future, look at the past. Caches were unknown in the 286 PC generation, and only became noticeable when the 386 moved up to 25MHz. The 486 expends half of its 1.2 million transistor budget on the MMU and cache system. This trend will continue.

Now that the cache has moved on-chip, the next step is the use of wider system data busses and burst-mode bus controllers to accelerate I/O. I/O includes disk, LAN, and graphics-controller traffic. Chip sets for this purpose have been introduced from several sources. (I wonder whether a personal computer really needs such devices. A 486-based microcomputer with several intelligent high-speed I/O controllers starts looking a lot like a mainframe.)

Data compression chips are already being incorporated into tape backup controllers, and will make a future appearance on your disk and LAN controllers. (Like caches, data compression chips provide a *statistical* performance improvement. Traditional DOSes are based on deterministic physical disk storage parameters. Everybody needs more disk storage.)

All of the above require an enormous up-front investment, which in turn requires a very large market. The highest semiconductor technology is now appearing first in the PC, not minicomputer, marketplace. This is another trend which will continue. PJ

REFERENCES

1. Smith, A.J. 1987. "Cache Memory Design: An Evolving Art," *IEEE Spectrum* (Dec.), pp. 40-44.
2. Patterson, D.A. and Hennessy, J. 1990. *Computer Architecture, a Quantitative Approach*, Morgan Kaufmann, pp. 408-432.
3. Stone, H.S. 1987. *High Performance Computer Architecture*, Addison-Wesley, pp. 29-69.
4. Smith, A.J. 1982. "Cache Memories," *ACM Computing Surveys*, Vol. 14, No. 3, pp. 473-530.
5. Milenkovic, M. 1990. "Microprocessor Memory Management Units," *IEEE Micro* (Apr.), pp. 70-85.

PAINT POWERFUL DATA-ENTRY SCREENS WITH POWER SCREEN™

Design screens and menus with PAINT™, placing input fields and messages precisely where you want them to appear in your final application. PAINT stores screen definitions in a single file — which can be modified without recompiling or relinking your application. Your program manages screen display and I/O through the POWER SCREEN *Runtime Library*. All essential features are supported, including:

◆ Block multi-field or single-field I/O ◆ Automatic range checking ◆ Configurable editing/menu key definitions ◆ Application context-sensitive, cross-referenced help ◆ Virtual/automatic scrolling screens within viewports ◆ Multiple, overlapping viewport display ◆ Plus much, much more!

The Runtime Library can be linked directly with Turbo Pascal applications, or installed as memory resident.

All this for just $149.00

Includes complete sample programs, a comprehensive reference manual, and the Norton Instant Access program and guides to assist you during program development. And, we offer a 30-day money back guarantee. Supports Microsoft C/QuickC, Turbo C, Turbo Pascal, QuickPascal and QuickBASIC.

Other powerful products from Blaise Computing:

C ASYNCH MANAGER™	$189
ASYNCH PLUS™	$189
View232™	$189
C TOOLS PLUS™	$149
Turbo C TOOLS™	$149
POWER TOOLS PLUS™	$149
POWER SEARCH™	$149

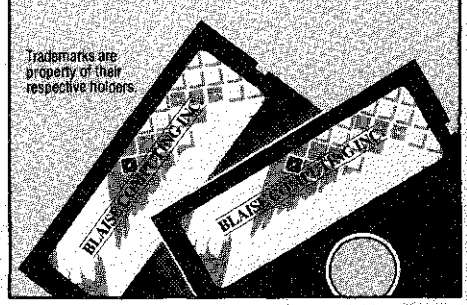
Call today for more information

(800) 333-8087 Fax (415) 540-1938

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316 Berkeley, CA 94710 (415) 540-5441

Trademarks are property of their respective holders.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Breaking the Rules Again

By Paul Barkley

In in the previous issue of *PJ* (8.4) I discussed various consulting dilemmas as seen from my bird's-eye view here in Washington, D.C. My locus operandi is fast becoming better known for its shady deals than its cherry trees, with everyone from park vendors to the mayor being heaped with attention they never asked for. My own experience as a consultant has convinced me that the best way to avoid trouble is to nip it in the bud, before it blossoms under the glare of unwanted scrutiny.

Figuring out how to steer clear of sticky situations is, of course, a mixture of personal style and hard-earned knowledge. If you're one of those unfortunates who can only learn from your own mistakes, may the force be with you, because you're going to need it. If, on the other hand, you don't mind benefitting from a little free advice, find a comfortable chair and we'll explore a few hypothetical predicaments and how to handle them.

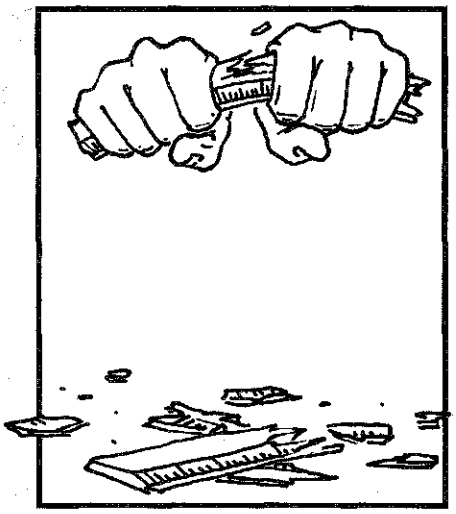


ILLUSTRATION BY JIM CARPENTER

MURKY ETHICS

I went over some thoughts about whether or not you go along with what a client wants when you know that their choice is technologically wrong in my last column. Now consider what you do when the client wants you to do something ethically wrong.

First, you have to decide whether what the client wants is outright fraudulence, simply bad business practice, or none of your business. Here's an example: what do you do when a client asks you to change an invoice to show more hours at a lower rate? The client is willing to pay the exact amount that you've billed, say 100 hours at $80 an hour, but has trouble justifying your high rates to his superiors. He wants an invoice for 200 hours at $40 an hour.

This is murky. Since you're billing the same total, the client's company is not paying any more for it. On the other hand, there must be a rule somewhere in the client's company that says consultants or subcontractors can only charge so much per hour. You're being asked to circumvent the rules of the organization in order to continue working for them.

It may be that it's only your client's immediate supervisor who doesn't like to pay high rates. In this case, you can make the argument that you owe your allegiance to your client, not to his or her boss.

You can also make the argument that, like a certain brand of hair coloring, you cost a lot but you're worth it. You're able to accomplish twice as much as the average consultant in the same amount of time, so you therefore charge twice as much an hour. If someone wants you to pretend that you worked twice as many hours at a "regular" consulting rate, why should you care?

The problem is that once you start fiddling with the hours, it's awfully easy to start fudging a little. A consultant I once knew, an eminently practical if not entirely upright man, once said that he didn't much care what rate he charged as long as he got to keep track of the hours. If the client wanted to pay $5 an hour, that was fine; he would just bill out 640 hours per week and everything would come out even. (Somehow I suspected that it might come out more than even.)

FUDGING AND AUDITS

So, should you adjust your billing rate and hours arbitrarily to suit a client? I might do it on a one-time basis, but I would try to find a way to get it straight for the next time. One of the biggest problem clients in this regard is the federal government. There are definite rules in some places about how much they can pay for consulting services. If they really want you, they may ask you to bend this rule for them by billing extra hours.

This is one place where I absolutely would not do so. The reason is simple: while small contracts are generally not audited, virtually all larger contracts are audited by a special organization of nasty people who always arrive on Friday afternoons or the day before Christmas, looking for people who have time cards that say they're present when they're really on their way to the beach or the ski slopes. The government is not very efficient, and it's not very smart, but it *is* large, and you don't want to get wrapped around its axle.

Somehow, I don't think that the auditors would be amused that your client told you it was okay to bill twice as many hours at half the price. The job of the auditors is to see to it that every hour billed to the government is an hour worked. How are you going to justify that week where you supposedly worked 640 hours?

SOME STICKY WICKETS

What do you do when you have a client that is consistently late in paying bills? The proper method is to issue monthly statements with interest charged on your unpaid invoices. One and a half percent per month is typical.

But if you do that, your client may get annoyed. Many clients have a history of paying all their bills late, and ignoring the interest charges. Bringing this matter to their attention can cause clients to look elsewhere for consulting

Paul A. Barkley can be reached at 13312 Neerwinder Place, Germantown, Maryland 20874.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

services. Some consultants simply compute the interest, figure out what that amount would be in consulting hours, and add those hours to the bill, estimating that the client will probably be as late on this bill as they were on the last one.

This is effective, but again I would caution that it's not a good practice. When you keep your own hours, as most consultants do, your practice is based on complete trust by the client. They expect that if you bill them for 6 hours, you have worked 6 hours. When you start to fudge, it's hard to stop. Why not give yourself a 10% raise while you're at it, without all the bother of telling your clients that your rates went up? Well hey, you can do this by adding 10% more hours to your bill. My suggestion? Don't.

UNDERESTIMATING

What about the opposite problem? Suppose you underestimated the amount of time something would take, and now you don't feel that you can charge the client for all the hours worked. You realize that you're going to have to eat some hours. How do you do it?

Many consultants are pretty sloppy about their timekeeping. They work late hours without writing them down and then send the client a pretty general bill. Although this might result in billing more hours than you should, more often than not it results in a cost-contained bill to the client and a lot of off-record hours for you.

Let me make this perfectly clear: NEVER DO THIS! Why? Because if you give someone something and they can't see it, you not only incur the loss of the gift, you also don't get any credit for it. You're not working in an O. Henry story, you know; if you worked until midnight all last week for a client, but only charged them for eight hours a day, you'd better show that uncompensated time on your bill somewhere. Let them know that you underestimated the amount of time, due to unforseen technical problems (never say the delay was because you're so dumb you couldn't guess how long it would take). But, you continue, because you feel bad about having told them it would only take a certain number of hours, you've decided to eat those hours and only charge the original estimate. Clients really like that sort of accounting. If you're going to do work for free, at least turn it into a marketing effort.

DRAWBACKS

There is one drawback to revealing uncompensated time. If you do it all the time, it will eventually occur to the client that if you can't estimate, chances are you're not too bright, and you may be causing other problems somewhere. So try to work few or no uncompensated hours, and try not to repeat the performance next month.

Another aspect to this uncompensated overtime business is that in the world of medium-size contracting, say awards of $10 million to $200 million, labor rates in bids have a profound effect on winning or losing. Although a few bidders may be excused for technically unresponsive proposals, most bidders who have been around a while make it past that first cut.

The final cut generally comes down to plain old dollars. The government, which is the prime buyer of consulting contracts in this price range, is not very smart about purchasing quality. The rules they must play by don't allow much slack for arguments that a particular vendor is so efficient that although the labor rates are 50% higher, they can do twice as much work in the same amount of time. That doesn't carry any weight, because it's almost impossible to document. Who can say one system analyst is, say, 19% more productive than another?

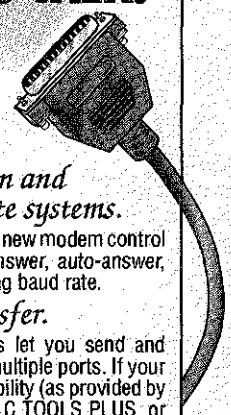
DOCUMENT TO WIN

There is a way, though, to show that your $25-an-hour person is worth more than someone else's $25-an-hour person. Document the fact that your people work more than 40 hours per week, with the excess as uncompensated overtime.

If your company charges $10 an hour for a technical writer, then you bill the client $400 a week (plus all the overhead stuff which we won't discuss). If you can show that this person regularly works 50 hours per week, then you can demonstrate that the government is getting a technical writer from you at an effective base rate of only $8 per hour, which is less than the $10 an hour your competition charges. If you can show

WE HAVE WAYS TO MAKE YOU TALK.

(And let you listen in, too)



with tools from Blaise Computing.

Talk to your modem and connect with remote systems.

Go online with ease with our new modem control routines. Initialize, dial, answer, auto-answer, and auto-adjust for incoming baud rate.

Let's talk file transfer.

XModem/YModem routines let you send and receive multiple files over multiple ports. If your program has ISR/TSR capability (as provided by our POWER TOOLS PLUS, C TOOLS PLUS, or Turbo C TOOLS), you can talk file transfer in the background while your foreground process talks data input, or database query, or whatever.

The file transfer capabilities include 1K packets, CRC error checking, YModem (multi-file transfers with file name and size preserved), automatic adaptation to incoming packet size and error detection method. And we're talking fast — all file transfer routines have been optimized for maximum throughput.

Let's talk basics.

C ASYNCH MANAGER™ and **ASYNCH PLUS™** retain the features which have made them the libraries of choice for asynch projects — buffered interrupt-driven input and output to multiple COM ports with speeds up to 19200 Baud, XON/XOFF protocol, hardware handshaking and much more.

C ASYNCH MANAGER supports Microsoft C, QuickC and Turbo C. ASYNCH PLUS supports Turbo Pascal and QuickPascal. Both products require DOS 2.00 or later and an IBM PC, XT, AT or PS/2 machine or true compatible.

Let's talk price.

Why pay more? Our ASYNCH packages are just $189, including complete source code, a comprehensive reference manual with extensive examples, sample programs and online help.

Turn that serial cable into a party line with View232!

Debugging serial communications doesn't have to leave you talking to yourself. **View232™** turns your computer into a serial data analyzer that lets you listen in on the conversation between any two serial devices — and this party line provides a transcript of what was said!

View232 displays the data as it flows in both directions. Save a whole transmission in a buffer, then browse through or search the buffer for a pattern or specific character. And you can save the data to disk or print it for later study.

VIEW232 is easy to use. And we supply the cable, all for just $189!

Let's talk track record.

Blaise Computing has produced a collection of tools that are unsurpassed for reliability, flexibility, and ease of use.

And, we're talking guaranteed.

If during the first 30 days you're not completely satisfied, we'll refund your money.

Call (800) 333-8087 today!

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 316
Berkeley, CA 94710
(415) 540-5441
FAX (415) 540-1938

Trademarks are property of their respective holders.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The military says it's not up to them to tell companies how to run their businesses. Right.

this for many or all of your labor categories, you'll probably win.

One argument I've heard against this says that anyone looking at those figures would conclude that your people are so incompetent that they have to work 50-hour weeks just to keep up. Fortunately, while that might even be true, it is not taken into consideration. Remember, I said the government isn't usually able to make allowances for quality differences. (Kind of makes you nervous when you know everything they're using was purchased from the low bidder, doesn't it?)

GREEN TIMING

There's a dark side to this, though. As far as I can tell, it's called "green timing." I've only heard it whispered around the water coolers, and it's always said in a bitter voice talking about how a competitor on a certain contract did it to the person in question.

Green timing apparently refers to the practice of encouraging your people to put down excess hours. They actually work 40 hours, you pay them for 40 hours, and you charge the government 40 hours. But they put down 50 (or whatever) hours on their time sheet, with the excess going to uncompensated overtime. Since the contracting employee knows that the government is not being charged for those hours, and since you've leaned on them to do this "little thing" for you, the employee doesn't worry about the morality of the situation. It's easy enough to do.

The problem, of course, is that this misleads the government into thinking they received more than they did. Sure, you gave them the same deliverables that you would have given them without the deception, but you can claim that the quality is higher because your people worked harder and longer. Hmmm.

My advice? Don't do it. The odds of getting caught on uncompensated overtime are probably very low. Experience has shown that when this is suspected, and audits are performed, employees return to recording actual numbers until the heat is off. Although it sounds somewhat harmless, it's an abuse of the numbers game, and it's wrong.

One company that I heard about has a reputation for not just encouraging their employees to write down the extra hours—they want them to work the

OPTIMAL LINKER TECHNOLOGY

That's right, SLR Systems is THE source for the fastest, most powerful linking technology available. Wherever you are headed, chances are we are there, or working on it. If you don't see what you need here, talk to us.

OPTLINK technology is designed to have no inherent limitations. There is NO internal limit on the number of modules, libraries, publics, segments, etc. The only limit is the amount of disk space available.

OPTLINK/Windows:

Speed: Faster than any other linker, up to 11 times faster than LINK4. Utilizes EMS/XMS before thrashing to disk.

Syntax compatible with LINK4.

Supports Codeview debugging information. Optionally include debug information only from specific modules.

Detailed segment mapping and a complete global cross reference.

Implements FIXDS algorithm to eliminate need for MakeProcInstance, FreeProcInstance, and EXPORTS with EXE files.

Includes OPTIMP, an import library builder. *Only $250*

Other linkers available include:

Standard **OPTLINK** Generate COM, SYS, and DOS EXE files. Only $150

OPTLINK/Compress Same as OPTLINK but can generate fast self-extract-to-run EXEs 25% to 50% smaller than EXEPACK. Ideal for conserving disk space. Only $250

OPTLINK/2 Complete segmented linker for 16-bit OS/2. Same as OPTLINK/Windows except this linker runs under protected mode OS/2. Only $250

OPTLINK/Locate Generates Intel HEX or OMF output, besides COM, SYS, and DOS EXE. Locate by class and/or segment. Phase for initialized data in rom. Publics and linenumbers in OMF output. Omit classes and/or segments from output. Located CV debugging information to separate file. Only $395

SLR SYSTEMS INC.

1622 N. Main Street, Butler, PA 16001 (412) 282-0864 Voice (412) 282-7965 FAX (412) 282-2799 BBS

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

hours, too. Well, this certainly takes care of the unethical part, but I think a strong argument can be made that when an employee comes to a company, if they're signing on for a 50-hour week, they ought to know that up front. Of course this is never mentioned, because otherwise candidates would use both hours and compensation in computing the relative value of employment offers. So it's probably an employee abuse situation. On the other hand, if you wind up working extra hours and your company can benefit from the lower effective rate, by all means record the time.

Interestingly enough, the Pentagon recently rejected arguments that it should forbid companies from *forcing* employees to work uncompensated overtime. The military says it's not up to them to tell companies how to run their businesses. Right.

MORE CUTE TRICKS

Another cute client trick: they ask you to purchase some product for them, such as add-on boards or software, and bill these products as hours worked instead of as products purchased. Again, if you don't go along, your client might find someone else who will. If you do go along, you incur the same risk that we talked about before when you start fudging hours.

Here's a suggestion: tell your client that you have no way to pass through that sort of stuff on your books. You have your books audited by an outside accounting firm and are prohibited from doing this because it masks your actual operations. Your invoices have to match your actual product (services). You'd like to help them out, but you just can't. You even tried to do this once before with another client who wanted the same thing, and it simply didn't work out.

This just might work to get you off the hook without completely offending the client. It also sends a quiet message to the client that since this is not healthy for your books, it's probably not healthy for their books, either.

PAPERWORK WOES

Your client is having trouble completing the paperwork necessary to get you started on a task, but the work must begin anyway. They assure you the paperwork will be forthcoming, and want you to start now. They also tell you to track your hours, but invoice them as if they were worked after the contract was actually in place.

This is a tough one. Again, you're fiddling with the hours. If you're certain that the client will come through, you can of course make them happy and get started. Last week I came across a person who started his group on some client work, only to find out a week later that the project subsequently had not been approved and they therefore could not bill out those hours. So sorry. Ask yourself how you'd feel if you had to eat a week's worth (or more) of work.

Last year a consultant told me that under considerable pressure he started a task for a firm before the contract was in place, and after work was well underway the client wanted to renegotiate lower rates or not pay him at all for the work already completed! Without a contract in place, the consultant was at a considerable disadvantage bargaining. He came to the conclusion that this is common practice for a number of firms.

TO SKEW OR NOT TO SKEW

Do you skew a study to suit your client contact and his or her personal constituency? Your first reaction is probably negative. How outrageous of a client to pressure you to abandon your professional judgement and integrity to advance their grubby little interests!

Suppose I said that the study only represented $20,000 out of a $10 million dollar contract? Are your antennae starting to quiver? Suppose the issue is close to a toss-up and you don't particularly care one way or the other. Skewing gets a little easier all the time, doesn't it?

You can cave in and perhaps no one will be the wiser. Personally, I find it alarming to think that a questionable act of mine will be nicely documented and available for anyone to study for years to come.

AARDVARK PROTECTION

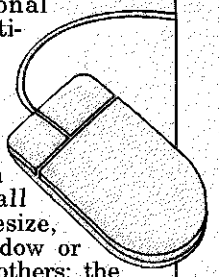
Hey, you're a clever consultant. You can do much better than just caving in. First, play CYA (loosely translated: put a saddle blanket over your donkey). Document all the assumptions necessary to

6.0 AND COUNTING!

Integrate sophisticated features into your Microsoft C and QuickC applications with

C TOOLS PLUS/6.0™

C TOOLS PLUS version 6.0 is filled with many advanced routines for developing high-powered C applications, including: virtual, stackable menus and windows with full mouse support and optional "drop shadows"; multiple virtual pop-up help screens; a miniature multi-line editor for gathering user responses in a robust fashion; a single function call which can move, resize, and promote a window or menu on top of all others; the ability to update covered windows automatically when they are written to; support for EGA, VGA, and MCGA text modes including 30-, 43-, and 50-line modes; support for the enhanced (101/102 key) keyboard.



All this and more for only $149!

C TOOLS PLUS/6.0 also contains functions for writing interrupt service routines; creating pop-up memory resident applications; general memory "peeks" and "pokes"; access to the DOS PRINT utility; as well as many other general utility functions and macros.

COMPLETE PROFESSIONAL PACKAGE.

Blaise Computing's function libraries offer easy to use solutions to your programming needs. You get source code, complete sample programs, and a comprehensive reference manual with extensive examples. Supports QuickC and Microsoft C 5.0 and later.

30 DAY GUARANTEE.

If during the first 30 days you are not completely satisfied, we'll refund your money.

Other powerful products from Blaise Computing

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

lead to the possibly erroneous conclusion that the client is asking for. Studies should always contain the conditions under which the conclusions will be valid, anyway.

For example, if your conclusion is that the earth is flat, you'd better note that *if* ships are never seen hull down (over the horizon), *if* there's an amazing waterfall at the edge of the ocean just east of the Bermuda Triangle, and *if* the Challenger brought back some great pictures of the turtle upon whose back the disc (not disk) of the world rests, then the world is most assuredly flat. If it turns out that some or all of those assumptions are false, then no one will blame you if the conclusion is false, too. (Note that you must claim never to have been anywhere near the Bermuda Triangle; it was slightly, but only slightly, outside the scope of the present task.)

GOOD ANALYTIC PRACTICE

I'm being a little facetious, but you can easily find more subtle assumptions that lead you to a particular conclusion. For example, a client-suggested LAN topology is assuredly based on assumptions about the number and type of workstations, frequency of access, and types of user applications. Clearly state these assumptions.

I might point out that, repugnant as the exercise may be, the philosophy of CYA (Cover Your Aardvark?) happens to agree with good analytic practice. When the client sees the report, the explicitly stated assumptions may get his or her attention.

Sometimes the client has jumped to the wrong conclusion, asked you to document and support it, and is then appalled to see what conditions would have to be true to reach said conclusion. By not refusing to participate, you may actually be able to affect the outcome positively. At the least you can do a professional job, keep the client happy, and prevent the client contact's internal enemies from skewering you for being an incompetent fool. Keep in mind that if anything goes wrong with a blatantly skewed study, you're the one who's going to catch it in the ear. I suggest you draw the line if the client, after reading your draft, asks you to omit certain assumptions that are necessary to reach what would otherwise be an unreasonable conclusion.

CHIP AWAY

Finessing your way through these kinds of problems requires good thinking, which is always tempered with a dose of foresight. Only you know what you can live with and what sorts of chances you're willing to take. Your work as a consultant gives you plenty of room in which to be creative. Just keep in mind that lots of people whose jobs would indicate that they're just as smart as you, and probably a good deal more ambitious, have been wrong about the amount of bluffing they can get away with. Like cherry blossoms at the end of spring, the chips have a way of falling where they may. PJ

OPUS SOFTWARE

"Without a doubt, OPUS Make is the hottest make utility on the market"
Tom Swan, PCWorld

OPUS Make

For professional programmers, OPUS Make is the superior choice in a make utility. It is the fastest most full-featured make utility there is. Features include:

- DOS version uses only 3K memory!
- Multiple directory support.
- Supports: Polytron PVCS™, Burton Systems TLIB™, Microsoft LIB™, SLR Systems OPTLIB™
- Generates automatic response files of unlimited length for LINK and LIB.
- OPUS MKMF included: An automatic dependency generator that fully understands C preprocessor directives.

Both DOS and OS/2 executables for only: $129 MC, Visa, COD, or PO. 60 day MBG!

Call 1-800-248-OPUS
International: 415-684-7861
1032 Irving Street, Suite 439, San Francisco, CA 94122
Trademarks owned by their respective companies

Circle 254 on Reader Service Card

Data Junction® converts data files!

Any ASCII • dBASE • R:BASE • Lotus 1-2-3
Excel • Paradox • Informix • Q&A • Oracle • Xdb
askSam • SC4 • Clarion • DIF • SYLK • Symphony
ACT! • Platinum • DAC • and other application files
IBM/Mips/Sparc/Cray/Vax/Intel/68000 Binaries
Btrieve • c-tree • EBCDIC • Packed data
. . . and more!

- No programming-completely menu driven!
- Select and rearrange records, fields and characters!
- Instant script file creation for total batch processing!
- Map new data structures automatically by name!
- Ideal for OEM applications!

". . . an invaluable integration tool." -Byte
". . . more than worth its price." -PC Magazine
". . . a surefire winner." -Computer Language

Standard $99 - dBASE, ASCII, 1-2-3, Q&A, Enable, DIF, askSam & merge files for WordPerfect, Word & WordStar
Professional $199 - Standard formats plus Binary/EBCDIC, SC4, SYLK Paradox, R:BASE, Clarion, Excel, c-tree & Btrieve
Advanced $299 - Standard & Professional formats plus Informix, Xdb, Oracle & C-ISAM

Call 800/444-1945 or 512/482-0824
MC/VISA • AMEX • COD • Corp. PO.
available for LANs, Xenix & Unix
Tools & Techniques Inc. • 1620 W 12 • Austin TX 78703

T&T TOOLS & TECHNIQUES

Circle 327 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Here's what they say about Zortech C++

" Zortech is a truly fine compiler...If you've been waiting for a major player to offer a professional C++ development system for OS/2 and Windows, as well as DOS, wait no longer... Zortech has it! "

Richard Hale Shaw, PC Magazine, p.38, March 13, 1990

" Zortech has done a commendable job with C++ 2.0 and I recommend it highly...The debugger is impressive...Get the Developers version...it's worth the money."

Bruce Eckel, Micro Comucopia, pp. 8-17, March 1990

" Zortech C++ is one of the best MS-DOS products I've had the luck to use....I can highly recommend the Zortech 2.0 release."

Scott Robert Ladd, Dr. Dobbs Journal, pp. 64-73, January 1990

" The Zortech compiler is a well-crafted product; in particular it is efficiently documented and it produces high quality object code. I can recommend it as a fine package"

Paul G Smith, .EXE Magazine, pp. 16-22, March 1990

AT&T™C++ V2 Specification
- ✓ *Multiple Inheritance*
- ✓ *Type Safe Linkage*
- ✓ *Pointers to Members*

Compiler Features
- ✓ *Native code compiler with separate global optimzer*
- ✓ *Improved MSC Source Level Compatibility*
- ✓ *MS Windows™ Compatible*
- ✓ *CodeView™ Compatible*
- ✓ *Fast Graphics Library with C++ interface*
- ✓ *Easy to use TSR functions*
- ✓ *Standard Library Source Code included with Developer's Edition*
- ✓ *Seamless LIM/EMS Support via new handle pointers or directly via EMS library functions.*
- ✓ *Full MS Mouse Library*
- ✓ *OS/2 Compiler Option*
- ✓ *99% ANSI C Compatible*
- ✓ *Improved code size/speed*

C++ Source Level Debugger
- ✓ *Also Debugs C*
- ✓ *Assembler Debugging with access to registers and memory.*
- ✓ *16 Debugging Windows*
- ✓ *Multiple Statement Lines*
- ✓ *Break/Trace/Watchpoints*
- ✓ *Dual Monitor Support*
- ✓ *Full C++ name unmangling for easy use*
- ✓ *Block memory write protect*

C++ Tools Classes
- ✓ *25 C++ Classes with full source code*
- ✓ *Includes new Text User Interface Classes*
- ✓ *Event Queue, BCD Maths, Linked Lists, Money, DOS error handling classes, text windows and editing classes, virtual arrays, time and date handling, directories and filenames, interupt vectors, etc...*

PRICES

C++ Compiler	$199.95
C++ Debugger	$149.95
C++ Tools	$149.95
Library Source	$149.95

Save $200 - Get the Developer's Edition for only $450 (includes all the above items).

OS/2 Option	$149.95
C++ Video	$499.95

USA: Zortech Inc.
4-C Gill Street
WOBURN MA01801
Voice: 617-646-6703
Fax: 617-643-7969

EUROPE: Zortech Ltd.
106-108 Powis Street
LONDON SE18 6LU
Voice: 44+ 81-316-7777
Fax: 44+ 81-316-4138

"ANNOUNCING V2.1"

640K Memory Barrier Smashed!

- **New** *VCM™ (Virtual Code Manager) technology*
- **New** *Rational DOS Extender technology for compiling/ debugging massive programs*
- **New** *Virtual C++ Source Level Debugger requires only 4k RAM!*
- **New** *Remote Debugging via serial port*
- **New** *Powerful Environment with Browser*
- **New** *Completely Revised & Expanded C++ Tools*
- **New** *Improved Compiler Optimization*

Zortech VCM™ for DOS

With Zortech's Virtual Code Manager (VCM) you can compile standard MS-DOS applications containing up to 4Mb of code. VCM is a sophisticated virtual memory system that dramatically improves performance over conventional overlay methods. Naturally, our debugger understands VCM too!

Rational™ DOS Extender Technology...

Version 2.1 incorporates this new technology for compiling and debugging really big programs on 286, 386 or 486 based PC's. You can also use V2.1 together with Rational Systems DOS Extender (purchased separately) to produce your own applications which can access memory beyond the 640k DOS limit.

C++ Debugger in 4k RAM!

Zortech's Virtual C++ Source Level Debugger can now locate itself in extended memory on 386 machines. This requires only 4K of conventional RAM!

STOP PRESS - NEWS FLASH

386 Compiler/Debugger Option (using Phar Lapp DOS Extender), UNIX 386 Compiler and OS/2 Debugger all available soon. Also new C++ Classes and Addison Wesley ZTC++ book.

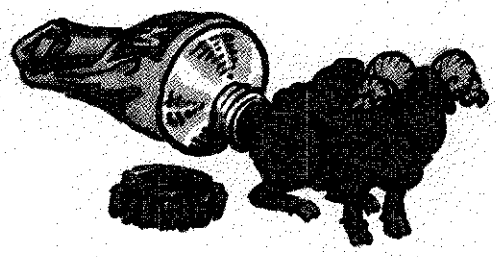
ORDER/UPGRADE HOTLINE 1-800-848-8408

Number 182 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Michael Abrash

More colors on the VGA



Exploring the complexity of palette RAM and the DAC

Installment 20: *In which we explore the obdurate complexities of VGA color, in the process becoming quite possibly the first computer publication ever to tap the wisdom of both Eurythmics and Yogi Bear at one sitting.*

Kevin Mangis wants to know about the VGA's 4-bit to 8-bit to 18-bit color translation. Mansur Loloyan would like to find out how to generate a look-up table containing 256 colors and how to change the default color palette. They are surely only the tip of the iceberg; hordes of screaming programmers are no doubt preparing to inundate the offices of *Programmer's Journal* with demands for a discussion of VGA color. As Eurythmics might say, who is this humble columnist to disagree?

I hope all of you know what you're getting into. To paraphrase Yogi, VGA is smarter (and more confusing) than the average board. There's the basic 8-bit to 18-bit translation, there's the EGA-compatible 4-bit to 6-bit translation, there's the 2- or 4-bit color paging register that's used to pad 6- or 4-bit pixel values out to 8 bits, and then there's 256-color mode. Fear not, it will all make sense in the end, but it may take us a couple of columns to get there.

I must begin by referring you to Michael Covington's excellent article, "Color Vision and the VGA," in the June/July 1990 issue of *PC Techniques*. Michael is an expert in many areas I know nothing about, including linguistics and artificial intelligence. Add to that list the topic of color perception, for his article beautifully describes the mechanisms by which we perceive color and ties that information to the capabilities of the VGA. After reading Michael's article, you'll understand what colors the VGA is capable of generating, and why.

Our topic this month complements Michael's article nicely. Where he focused on color perception, we'll focus on color generation; that is, the ways in which the VGA can be programmed to generate those colors that lie within its capabilities. To find out why a VGA can't generate a red as pure as an LED, read Michael's article. If you want to find out how to flip between 16 different sets of 16 colors, though, don't touch that dial!

I would be remiss if I didn't mention two more articles, these in the July 1990 issue of *Dr. Dobb's Journal*. "Super VGA Programming" by Chris Howard provides a good deal of useful information about SuperVGA chipsets, modes, and programming. "Circles and the Digital Differential Analyzer" by Tim Paterson is the article about fast circle drawing to which I referred in my previous installment.

VGA COLOR BASICS

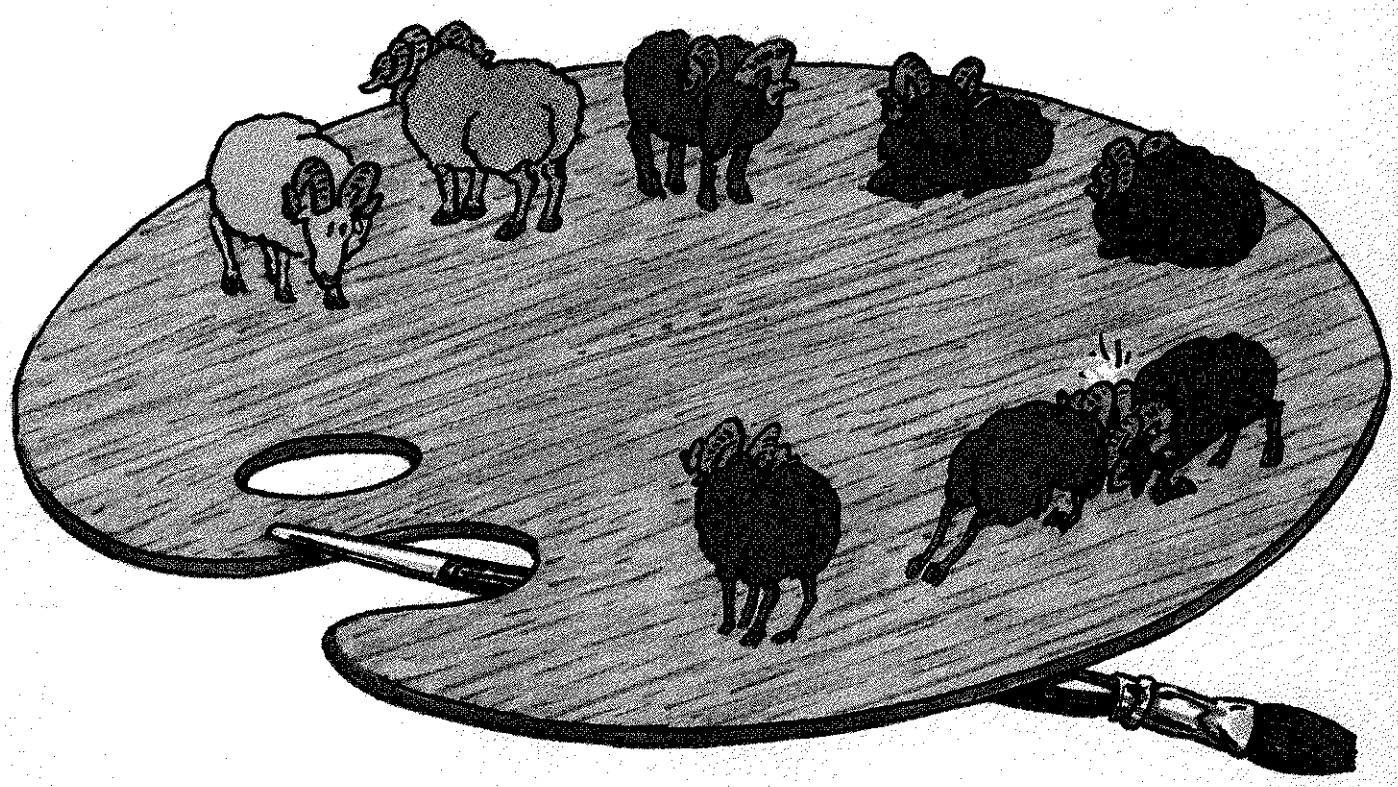
Briefly put, the VGA color translation circuitry takes in one 4- or 8-bit pixel value at a time and translates it into three 6-bit values, one each of red, green, and blue, that are converted to corresponding analog levels and sent to the monitor. Seems simple enough, doesn't it? Unfortunately, nothing is ever that simple on the VGA, and color translation is no exception.

THE PALETTE RAM

The color path in the VGA involves two stages, as shown in Figure 1. The first stage fetches a 4-bit pixel from display memory and feeds it into the EGA-compatible palette RAM (so called because it is functionally equivalent to the palette RAM color translation circuitry of the EGA), which translates it into a 6-bit value and sends it on to the DAC (Digital/Analog Converter). The translation involves nothing more complex than using the 4-bit value of a pixel as the address of one of the 16 palette RAM registers; a pixel value of 0 selects the contents of palette RAM register 0, a pixel value of 1 selects register 1, and so on. Each palette RAM register stores 6 bits, so each time a palette RAM

Contributing Editor Michael Abrash works for Metagraphics of Scotts Valley, California as a senior engineer. He is the author of Power Graphics Programming *and* Zen of Assembly Language. *Readers may write to Michael at P.O. Box 414, Burlington, VT 05402-0414, or contact him via MCI Mail, user name* MABRASH.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



register is selected by an incoming 4-bit pixel value, 6 bits of information are sent out by the palette RAM. (The operation of the palette RAM was described in installment #9 of "On Graphics," in *PJ* 6.6, November 1988.)

The process is much the same in text mode, except that in text mode each 4-bit pixel value is generated based on the character's font pattern and attribute. In 256-color mode, the palette RAM is not a factor from the programmer's perspective and should be left alone.

THE DAC

Once the EGA-compatible palette RAM has fulfilled its karma and performed 4-bit to 6-bit translation on a pixel, the resulting value is sent to the DAC. The DAC performs an 8-bit to 18-bit conversion in much the same manner as the palette RAM, then converts the 18-bit result to analog red, green, and blue signals, and sends the three analog signals to the monitor. While the DAC is a separate chip, external to the VGA chip, it's an integral part of the VGA standard and is present on every VGA.

(I'd like to point out that you can't speak of "color" in the color translation process until the output stage of the DAC. The 4-bit pixel values in memory, 6-bit values in the palette RAM, and 8-bit values sent to the DAC are all attributes, not colors, because they're subject to translation by a later stage. For example, a pixel with a 4-bit value of 0 isn't black, it's attribute 0. It will be translated to 3Fh if palette RAM register 0 is set to 3Fh, but that's not the color white, just another attribute. The value 3Fh coming into the DAC isn't white either, and if the value stored in DAC register 63 is red=7, green=0, and blue=0, the actual color displayed for that pixel that was 0 in display memory will be dim red. It isn't color until the DAC says it's color.)

The DAC contains 256 18-bit storage registers, used to translate one of 256 possible 8-bit values into one of 256K 18-bit values. The 18-bit values are composed of three 6-bit values, one each for red, green, and blue; for each color component, the higher the number, the brighter the color, with 0 turning that color off in the pixel and 63 (3Fh) making that color maximum brightness.

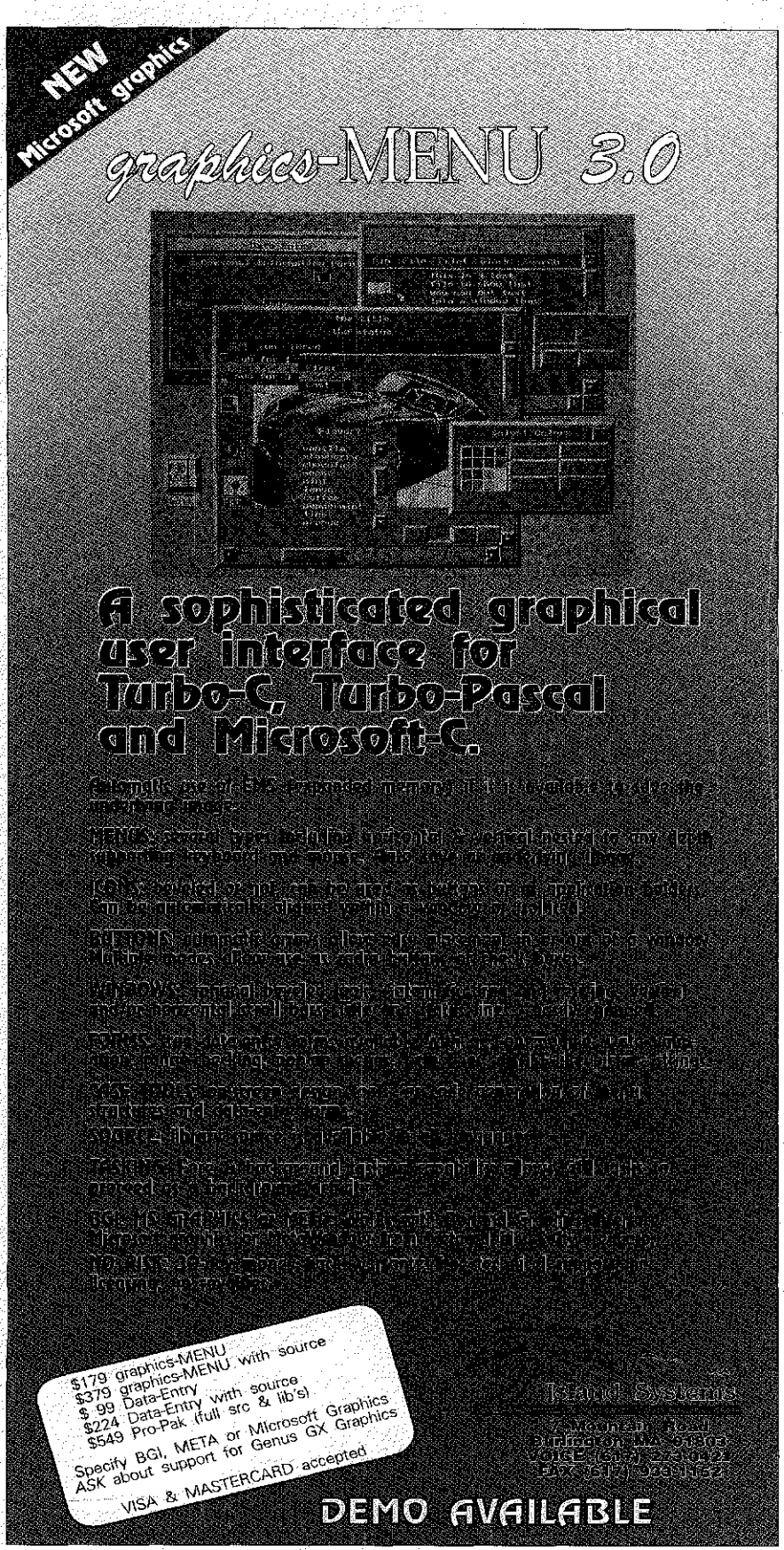
COLOR PAGING

"Wait a minute," you say bemusedly. "Aren't you missing some bits between the palette RAM and the DAC?" Indeed I am. The palette RAM puts out 6 bits at a time, and the DAC takes in 8 bits at a time. The two missing bits—bits 6 and 7 going into the DAC—are supplied by bits 2 and 3 of the Color Select register (Attribute Controller register 14h). This has intriguing implications. In 16-color modes, pixel data can select only one of 16 attributes, which the EGA palette RAM translates into one of 64 attributes. Normally, those 64 attributes look up colors from registers 0 through 63 in the DAC, because bits 2 and 3 of the Color Select register are both zero. By changing the Color Select register, however, one of three other 64-color sets can be selected instantly. I'll refer to the process of flipping through color sets in this manner as color paging.

While interesting, this seems somewhat half-baked; why bother expanding 16 attributes to 64 attributes before looking up the colors in the DAC? What we'd really like is to map the 16 attributes straight through the palette RAM without changing them and supply the upper 4 bits going to the DAC from a register, giving us 16 color pages. As it happens, all we have to do to make that happen is set bit 7 of the Attribute Controller Mode register (register 10h) to 1. Once that's done, bits 0 through 3 of the Color Select register go straight to bits 4 through 7 of the DAC, and only bits 0 through 3 coming out of the palette RAM are used; bits 4 and 5 from the palette RAM are ignored.

In this mode, the palette RAM effectively contains 4-bit rather than 6-bit registers, but that's no problem because the palette RAM will be programmed to pass pixel values through unchanged by having register 0 set to 0, register 1 set to 1, and so on, a configuration in which the upper two bits of all the palette RAM

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



registers are the same (zero) and therefore irrelevant. As a matter of fact, you'll generally want to set the palette RAM to this pass-through state when working with VGA color, whether you're using color paging or not.

Setting the palette RAM to a pass-through state is a good idea because the palette RAM is programmed by the BIOS to EGA-compatible settings, and the first 64 DAC registers are programmed to emulate the 64 colors that an EGA can display during mode sets for 16-color modes. This is done for compatibility with EGA programs, and it's useless if you're going to tinker with the VGA's colors.

As a VGA programmer, you want to take a 4-bit pixel value and turn it into an 18-bit RGB value. You can do that without any help from the palette RAM, and setting the palette RAM to pass-through values effectively takes it out of the circuit and simplifies life something wonderful. The palette RAM exists solely for EGA compatibility, and serves no useful purpose that I know of for VGA-only color programming.

256-COLOR MODE

So far I've spoken only of 16-color modes; what of 256-color mode? The rule in 256-color mode is: *don't tinker with the EGA palette*. Period. You can select any colors you want by reprogramming the DAC, and there's no guarantee as to what will happen if you mess around with the palette RAM. There's no benefit that I know of to changing the palette RAM in 256-color mode, and the effect may vary from VGA to VGA.

On the other hand, feel free to alter the DAC settings to your heart's content in 256-color mode, all the more so because this is the only mode in which all 256 DAC settings can be displayed simultaneously. By the way, the Color Select register and bit 7 of the Attribute Controller Mode register are ignored in 256-color mode; all 8 bits sent from the VGA chip to the DAC come from display memory. Therefore, there is no color paging in 256-color mode. Of course, that makes sense given that all 256 DAC registers are always in use in 256-color mode.

The palette RAM can be programmed either directly or through BIOS inter-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Use the BIOS unless there's a clear reason not to do so.

rupt 10h, function 10h. I strongly recommend using the BIOS interrupt; a clone BIOS may mask incompatibilities with genuine IBM silicon. Such incompatibilities could include anything from flicker to trashing the palette RAM; they may not exist at all, but why find out the hard way? My policy is to use the BIOS unless there's a clear reason not to do so, and there's no such reason that I know of in this case.

When programming specifically for the VGA you need to load the palette RAM only once, to store the pass-through values 0 through 15 in palette RAM registers 0 through 15. Setting the entire palette RAM is accomplished easily enough with subfunction 2 (AL=2) of function 10h (AH=10h) of interrupt 10h. A single call to this subfunction sets all 16 palette RAM registers (and the Overscan register) from a block of 17 bytes pointed to by ES:DX. ES:DX points to the value for register 0, ES:DX+1 points to the value for register 1, and likewise on up to ES:DX+16, which points to the overscan value. The palette RAM registers store 6 bits each, so only the lower 6 bits of each of the first 16 bytes in the 17-byte block are significant. (The Overscan register is an 8-bit register, about which we'll speak further another time.)

Alternatively, any one palette RAM register can be set via subfunction 0 (AL=0) of function 10h (AH=10h) of interrupt 10h. For this subfunction, BL contains the number of the palette RAM register to set and the lower 6 bits of BH contain the value to which to set that register. Having said that, let's leave the palette RAM behind (presumably in a pass-through state) and move on to the DAC, which is the right place to do color translation on the VGA.

SETTING THE DAC

Like the palette RAM, the DAC registers can be set either directly or through the BIOS. Again, the BIOS should be used whenever possible, but there are a few complications here. My experience is that varying degrees of flicker and screen bounce occur on many VGAs when a large block of DAC registers is set through the BIOS. That's not a problem when the DAC is loaded just once and then left that way, as is the case in Listing 1, but it can be a serious problem when the color set is changed rapidly ("cycled") to produce on-screen effects such as rippling colors. My somewhat limited experience is that it's necessary

ProGraphx

ProGraphx Menus™

Simply add four lines of code to your program and you can have a pull-down or pull-up menu system. Define a menubar with a text file, compile it and receive events calls back to your program. Change the menubar without recompiling your program to create demo versions or other adaptations. A sample application of a Microcoded processor is included along with its two editors.

ProGraphx Menus™	$99
Menus with Source	199

ProGraphx Toolbox™

Over 50 hi-speed graphics routines for EGA/VGA 16 color modes, including proprietary icon routines to display color images.

ProGraphx Image Editor, a complete color icon paint package to edit and view up to 50 icons at once. Simply paint an icon, then use the ***ProGraphx Toolbox™*** routines to display it with amazing speed. Over **500** sample icons for use in your own programs. ***Operation:Airlift***, a full screen scrolling arcade style game with source code.

All routines written in assembly for fastest possible performance.

ProGraphx ToolBox™	$199
Toolbox with Source	399

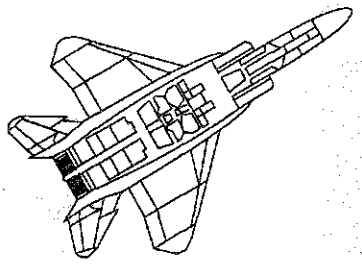
Stanwood Associates, Inc.
303 East Ohio Street Chicago, IL 60611

1-(800) 752-7982

(312) 828-9734 VISA/MC FAX (312) 828-9742

ProGraphx
Speed, Simplicity, and Power for Graphics Programmers

Compatible with Turbo C or Pascal, Zortech C/C++, Microsoft C or any language compatible with 80x86 object code. Uses less than 10K of your code segment!



Also available from :
Programmer's Shop &
Programmer's Connection

Circle 330 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
; Program to demonstrate use of the DAC registers by selecting a
; smoothly contiguous set of 256 colors, then filling the screen
; with concentric diamonds in all 256 colors so that they blend
; into one another to form a continuum of color.
;
        .model small
        .stack  200h
        .data

; Table used to set all 256 DAC entries.
;
; Table format:
;       Byte 0: DAC register 0 red value
;       Byte 1: DAC register 0 green value
;       Byte 2: DAC register 0 blue value
;       Byte 3: DAC register 1 red value
;       Byte 4: DAC register 1 green value
;       Byte 5: DAC register 1 blue value
;                       :
;       Byte 765: DAC register 255 red value
;       Byte 766: DAC register 255 green value
;       Byte 767: DAC register 255 blue value

ColorTable      label   byte

; The first 64 entries are increasingly dim pure green.
X=0
        REPT    64
        db      0,63-X,0
X=X+1
        ENDM

; The next 64 entries are increasingly strong pure blue.
X=0
        REPT    64
        db      0,0,X
X=X+1
        ENDM

; The next 64 entries fade through violet to red.
X=0
        REPT    64
        db      X,0,63-X
X=X+1
        ENDM

; The last 64 entries are increasingly dim pure red.
X=0
        REPT    64
        db      63-X,0,0
X=X+1
        ENDM

        .code
Start:
        mov     ax,0013h        ;AH=0 selects set mode function,
                                ;  AL=13h selects 320x200 256-color mode
        int     10h
                                ;load the DAC registers with the
                                ;  color settings
        mov     ax,@data        ;point ES to the default data segment
        mov     es,ax
        mov     dx,offset ColorTable
                                ;point ES:DX to the start of the
                                ;  block of RGB three-byte values
                                ;  to load into the DAC registers
        mov     ax,1012h        ;AH=10h selects set color function,
                                ;  AL=12h selects set block of DAC
                                ;  registers subfunction
        sub     bx,bx           ;load the block of registers
                                ;  starting at DAC register #0
        mov     cx,100h         ;set all 256 registers
        int     10h             ;load the DAC registers
                                ;now fill the screen with concentric
                                ;  diamonds in all 256 color attributes

        mov     ax,0a000h       ;point DS to the display memory segment
        mov     ds,ax
                                ;draw diagonal lines in the upper
                                ;  left quarter of the screen
        mov     al,2            ;start with color attribute #2
        mov     ah,-1           ;cycle down through the colors
        mov     bx,320          ;draw top to bottom (distance from
                                ;  one line to the next)
        mov     dx,160          ;width of rectangle
        mov     si,100          ;height of rectangle
        sub     di,di           ;start at (0,0)
        mov     bp,1            ;draw left to right (distance from
                                ;  one column to the next)
        call    FillBlock       ;draw it
                                ;draw diagonal lines in the upper
                                ;  right quarter of the screen
        mov     al,2            ;start with color attribute #2
        mov     ah,-1           ;cycle down through the colors
        mov     bx,320          ;draw top to bottom (distance from
                                ;  one line to the next)
        mov     dx,160          ;width of rectangle
        mov     si,100          ;height of rectangle
        mov     di,319          ;start at (319,0)
        mov     bp,-1           ;draw right to left (distance from
                                ;  one column to the next)
        call    FillBlock       ;draw it
                                ;draw diagonal lines in the lower
                                ;  left quarter of the screen
        mov     al,2            ;start with color attribute #2
        mov     ah,-1           ;cycle down through the colors
        mov     bx,-320         ;draw bottom to top (distance from
                                ;  one line to the next)
        mov     dx,160          ;width of rectangle
        mov     si,100          ;height of rectangle
        mov     di,199*320      ;start at (0,199)
        mov     bp,1            ;draw left to right (distance from
                                ;  one column to the next)
        call    FillBlock       ;draw it
                                ;draw diagonal lines in the lower
                                ;  right quarter of the screen
        mov     al,2            ;start with color attribute #2
        mov     ah,-1           ;cycle down through the colors
        mov     bx,-320         ;draw bottom to top (distance from
                                ;  one line to the next)
        mov     dx,160          ;width of rectangle
        mov     si,100          ;height of rectangle
        mov     di,199*320+319  ;start at (319,199)
        mov     bp,-1           ;draw right to left (distance from
                                ;  one column to the next)
        call    FillBlock       ;draw it

        mov     ah,1            ;wait for a key
        int     21h
        mov     ax,0003h        ;return to text mode
        int     10h
        mov     ah,4ch          ;done--return to DOS
        int     21h

; Fills the specified rectangular area of the screen with diagonal
; lines.
;
; Input:
;       AL = initial attribute with which to draw
;       AH = amount by which to advance the attribute from
;               one pixel to the next
;       BX = distance to advance from one pixel to the next
;       DX = width of rectangle to fill
;       SI = height of rectangle to fill
;       DS:DI = screen address of first pixel to draw
;       BP = offset from the start of one column to the start of
;               the next

FillBlock:
FillHorzLoop:
        push    di              ;preserve pointer to top of column
        push    ax              ;preserve initial attribute
        mov     cx,si           ;column height
FillVertLoop:
        mov     [di],al         ;set the pixel
        add     di,bx           ;point to the next row in the column
        add     al,ah           ;advance the attribute
        loop    FillVertLoop
        pop     ax              ;restore initial attribute
        add     al,ah           ;advance to the next attribute to
                                ;  start the next column
        pop     di              ;retrieve pointer to top of column
        add     di,bp           ;point to next column
        dec     dx              ;have we done all columns?
        jnz     FillHorzLoop    ;no, do the next column
        ret
        end     Start
```

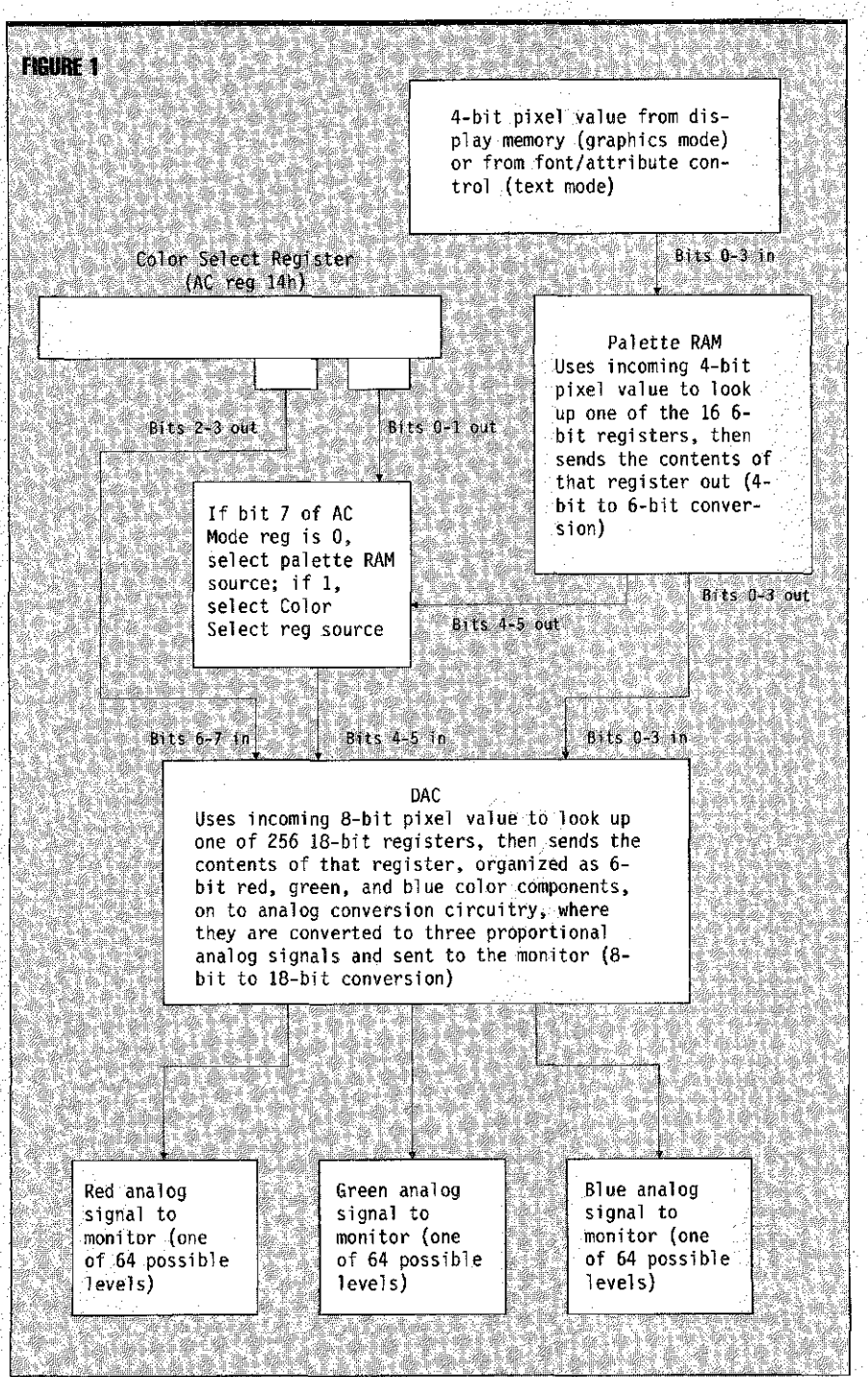

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

to program the DAC directly in order to cycle colors cleanly, although input from readers who have worked extensively with VGA color is welcome.

At any rate, the code in this article will use the BIOS to set the DAC, so I'll describe the BIOS DAC-setting functions next. Later, I'll briefly describe how to set both the palette RAM and DAC registers directly, and I'll return to the topic in detail in the next installment when we discuss color cycling.

Individual DAC registers are set by interrupt 10h, function 10h (AH=10), subfunction 10h (AL=10h), with BX indicating the register to be set and the color to which that register is to be set stored in DH (6-bit red component), CH

FIGURE 1

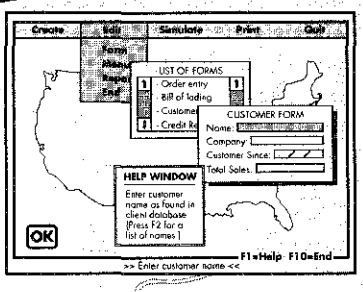

4-bit pixel value from display memory (graphics mode) or from font/attribute control (text mode)
Color Select Register (AC reg 14h)
Bits 0-3 in
Palette RAM
Uses incoming 4-bit pixel value to look up one of the 16 6-bit registers, then sends the contents of that register out (4-bit to 6-bit conversion)
Bits 2-3 out
Bits 0-1 out
If bit 7 of AC Mode reg is 0, select palette RAM source; if 1, select Color Select reg source
Bits 0-3 out
Bits 4-5 out
Bits 6-7 in
Bits 4-5 in
Bits 0-3 in
DAC
Uses incoming 8-bit pixel value to look up one of 256 18-bit registers, then sends the contents of that register, organized as 6-bit red, green, and blue color components, on to analog conversion circuitry, where they are converted to three proportional analog signals and sent to the monitor (8-bit to 18-bit conversion)
Red analog signal to monitor (one of 64 possible levels)
Green analog signal to monitor (one of 64 possible levels)
Blue analog signal to monitor (one of 64 possible levels)

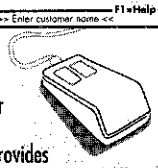
HI-SCREEN XL™

THE PROVEN USER-INTERFACE SOLUTION FOR YOUR PROGRAMS

Since 1983, Softway has specialized in user-interface development tools aimed at demanding professional programmers.

WINDOWS • MENUS DATA ENTRY • HELP





HI-SCREEN XL offers a high-level modular approach to screen management that provides for superior programming **control**, **flexibility** and **speed.**

- The built-in **screen editor** is used to create screens/windows/menus along with all field descriptions. Any **Hercules**, **CGA**, **EGA** or **VGA** graphics can also be captured. Screens are then saved in a compressed file format outside your program.
- **High-level calls** from your program give you total control over the user interface. Combine text and graphics, dialog boxes, context-sensitive help, elaborate data entry & menu management, keyboard & mouse control, etc.

WITH MOUSE AND GRAPHICS SUPPORT

- Supports all versions of **C**, **Basic**, **Cobol**, **Fortran**, **Pascal**, **dBase**, **Assembly**, etc.
- Use linkable (with C and MS Basic) or removable TSR screen driver (30-40K).

HI-SCREEN XL **$149**
HI-SCREEN XL *Professional Series*
(with mouse & graphics support): **$325**

30 day money-back guarantee • No royalties

Call today: **1-800-338-2852**
Tel.: 415-896-0708 • Fax: 415-896-0709

SOFTWAY, Inc.
185 Berry Street, Suite 5411, San Francisco, CA 94107

Circle **110** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

(6-bit green component), and CL (6-bit blue component).

A block of sequential DAC registers ranging in size from one register up to all 256 can be set via subfunction 12h (AL=12h) of interrupt 10h, function 10h (AH=10h). In this case, BX contains the number of the first register to set, CX contains the number of registers to set, and ES:DX contains the address of a table of color entries to which DAC registers BX through BX+CX-1 are to be set. The color entry for each DAC register consists of three bytes; the first byte is a 6-bit red component, the second byte is a 6-bit green component, and the third byte is a 6-bit blue component, as illustrated by Listing 1.

WHO YOU GONNA CALL?

Although the palette RAM and DAC registers should be set through the BIOS whenever possible, there are times when the BIOS is not the best choice or even a choice at all; for example, a protected-mode program may not have access to the BIOS. Also, as mentioned earlier, it may be necessary to program the DAC directly when performing color cycling. Therefore, I'll briefly describe how to set the palette RAM and DAC registers directly. Next time I'll discuss programming the DAC directly in more detail.

The palette RAM registers are Attribute Controller registers 0 through 15. They are set by first reading the Input Status 1 register (at 3DAh in color mode or 3BAh in monochrome mode) to reset the Attribute Controller toggle to index mode, then loading the Attribute Controller Index register (at 3C0h) with the number (0 through 15) of the register to be loaded. Do not set bit 5 of the Index register to 1, as you normally would, but rather set bit 5 to 0. Setting bit 5 to 0 allows values to be written to the palette RAM registers, but it also causes the screen to blank, so you should wait for the start of vertical retrace before loading palette RAM registers if you don't want the screen to flicker. (See why it's easier to go through the BIOS?)

Now you can write the desired register value to 3C0h, which has toggled to become the Attribute Controller Data register. Write any desired number of additional register number/register data pairs to 3C0h, then write 20h to 3C0h to unblank the screen.

The process of loading the palette RAM registers depends heavily on the proper sequence being followed; if the Attribute Controller Index register or index/data toggle data gets changed in the middle of the loading process, you'll probably end up with a hideous display, or no display at all. Consequently, for maximum safety you may want to disable interrupts while you load the palette RAM, to prevent any sort of interference from a TSR or the like that alters the state of the Attribute Controller in the middle of the loading sequence.

The DAC registers are set by writing the number of the first register to set to the DAC Write Index register at 3C8h, then writing three bytes—the 6-bit red component, the 6-bit green component, and the 6-bit blue component, in that order—to the DAC Data register at 3C9h. The DAC Write Index register

Professional software made affordable.

Sandpaper's System Debugger

The power of the 386 is yours for the taking.
System Debugger uses the 386 to create the ideal DOS development environment.
It uses the 386's protection, page tables and debug registers to provide the power of an expensive In-Circuit Emulator at a fraction of the price.

With System Debugger you'll ...
Set breakpoints on memory location and range reads and/or writes and **discover.**
⇒*When is the InDos flag incremented?*
⇒*What subroutine is overwriting my code?*
⇒*When does my malloc() space get corrupted?*
⇒*Am I overwriting DOS?*
Set breakpoints in ROM and **learn.**
⇒*Why does my software work in RAM but not in ROM?*
⇒*How does the BIOS work?*
Set breakpoints on I/O ports and interrupts and **know.**
⇒*Is a TSR conflicting with my software?*

Order now at no risk.
Money-back guarantee.

$99

Here is just a sampling of its features:
- Has symbolic debugging with source code of programs, Device Drivers, TSRs and ISRs.
- Runs standalone as well as in tandem with your current debugger.
- Doesn't slow down the system: load it by autoexec and it's ready when you need it.
- Stays active even if your program is hung with interrupts disabled.
- Displays undocumented DOS structures and memory map.
- Automatically comments interrupt calls so you don't need to look them up.
- Has an easy to learn intuitive windowed user interface.
- Finds those elusive annoying bugs which only exist in real time.
- Runs under DOS on 386SX/386/486 AT compatibles with 256K extended memory.
- Runs entirely in extended memory so that 640K is left where you need it.

To order, Call or FAX (818) 705-2657, or send a check to:

Sandpaper Software

P.O.Box 18935, Encino, CA 91416-8935

Please specify diskette format. CA residents please add sales tax.

Circle 32 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Develop Powerful Image-based Applications Quickly and Easily

with

VICTOR Image Processing Library

Powerful Image Processing

VICTOR simplifies development of scientific imaging, quality control, security, and image database software. Victor gives you powerful image processing (with functions like sharpen, outline, linearize, matrix convolution, overlay resize, flip, invert, mirror), screen display, and file handling capabilities.

Image-based applications can be developed in QuickC and Turbo C environments.

Gray Scale and Color Images

Victor allows you to handle 8-bit color and gray scale images from any source as TIFF/PCX/GIF files and display single or multiple images on EGA/VGA up to 800x600x256. Your software can have features such as: live video on VGA (frame grabber required), zoom, process and display multiple images, pixel editing and image processing. And, to get you up and running, we include Zip Image Processing software for rapid testing and prototyping of image processing and display functions. All processing functions are hardware independent. Graphics adapters supported but not required.

Victor supports Microsoft C, QuickC, and TurboC, includes demonstration and prototyping software, and full documentation. Source code available. Victor Library $195.

Your application will have powerful image handling capabilities.

Call (314) 962-7833 to order

VISA/MC/COD

Catenary Systems 470 Belleview St Louis MO 63119 (314) 9627833

Circle **321** on Reader Service Card

MORE 34010 Video Co-Processors from PC Tech

- Mono 736 x 1024 - 2048 x 1536
- Color 1024 x 768 (1280 x 1024 soon!)
- 512K - 4M DRAM
- Optional CCITT & DGIS support
- **AGE XoftWare X Window Server certified board**
- C Compiler, Assembler, Linker, Debugger, DGIS Development Kit & TIGA Toolkit available
- **TIGA** included with all boards
- Custom designs available
- **PC Tech understands the 34010**

For further information, please contact:

PC Tech

907 N. 6th St.
Lake City, MN 55041
(612) 345-4555 (Voice)
(612) 345-5514 (FAX)

PC Tech is a trademark of PC Tech, Inc.

PC, XT, AT, DGIS, XoftWare, and DGIS are trademarks or registered trademarks of their respective companies.

Circle **208** on Reader Service Card

GRAPHICS!

G·E·N·U·S MICROPROGRAMMING

INTRODUCING THE GX DEVELOPMENT SERIES

Hand coded in Assembly Language, these toolkits are designed to work alone or together with no code overhead. They support 8 languages and over 30 resolutions up to SuperVGA, for the fastest, tightest code. Save yourself time and money with our excellent tools and support.

An Exceptional Product Prog. Journal, 7/89

Powerful Image Graphics DBAdvisor, 1/89

Intuitive and Useful BYTE, 9/89

Well worth the investment Dr. Dobb's, 8/89

Excellent set of Libraries Comp. Language, 9/89

New Product! **GX Graphics** **$149**

A complete graphics library supporting all graphics primitives from *grPutPixel* to *grDrawEllipse*. Full support for logical operations (including rubber-banding), clipping, and drawing to off-screen virtual buffers in conventional memory or expanded (LIM 4.0). The cornerstone of the GX Series, use this graphics library instead of the BGI or MS libraries and make your program faster, smaller, and more portable across compilers–while supporting more modes.

New Version! **PCX Programmer's Toolkit 4.0** **$249**

The powerful and popular PCX toolkit is now even better! Over a dozen new functions have been added, more display adapters and compilers, and optimized assembly language for more speed and less code. Now a part of the GX Series, new features include image sizing and scaling, access to low level encode and decode, enhanced virtual buffer support, dot matrix /laserjet printing of scanned images, more examples (including real-time animation), plus all of the previous features that make the toolkit so popular. If you need PCX support, you need this.

Effects! **PCX Effects** **$99**

Add Special Effects to your PCX Toolkit Applications. Fades, wipes, animation, and music.

Fonts! **PCX Text** **$149**

Blazing graphics text in 80 different fonts, with bolding, underlining, etc.

All packages support 12 compilers for C, Pascal, Basic, Fortran, Assembly, Modula-2, Prolog, and Clipper. All modes of the Hercules, CGA, EGA, MCGA, VGA, and Super VGA adapters are supported, up through 800x600x256 (30+ modes in all). Assembly Language source code is optionally available. Trademarks are property of their respective holders.

NO ROYALTIES! 30-DAY GUARANTEE! SOURCE AVAILABLE!

Order Your Copy Today!

1-800-227-0918

COD CHK PO

11315 Meadow Lake
Houston, TX 77077
(713) 870-0737

FAX: (713) 870-0288
BBS: (713) 266-9362

MEMBER Better Business Bureau

MEMBER Software Publishers Association

Circle **261** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

then autoincrements, so if you write another three-byte RGB value to the DAC Data register, it'll go to the next DAC register, and so on indefinitely. You can also set all 256 registers by sending 256*3 = 768 bytes to the DAC Data register.

Loading the DAC is as sequence-dependent and potentially susceptible to interference as is loading the palette, so my personal inclination is to go through the whole process of disabling interrupts, loading the DAC Write Index, and writing a three-byte RGB value separately for each DAC register. Although that doesn't take advantage of the autoincrementing feature, it seems to me to be least susceptible to outside influences. (It would be even better to disable interrupts for the entire duration of DAC register loading, but that's much too long a time to leave interrupts off.) However, I have no hard evidence to offer in support of my conservative approach to setting the DAC, just an uneasy feeling, so I'd be most interested in hearing from any readers who have more concrete experience.

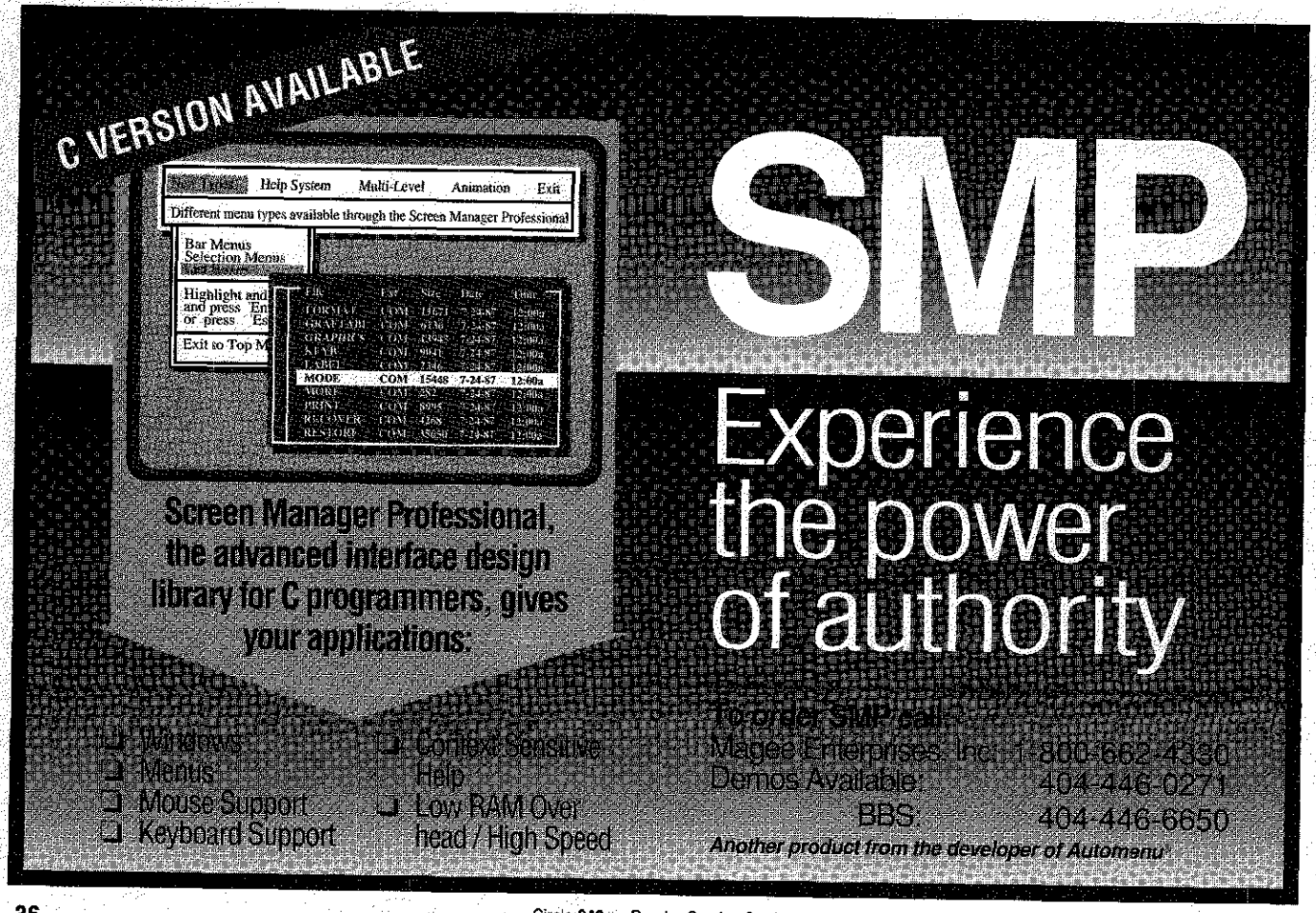
A final point is that the process of loading both the palette RAM and DAC registers involves performing multiple OUTs to the same register. Many people whose opinions I respect recommend delaying between I/O accesses to the same port by performing a JMP $+2 (this flushes the prefetch queue and forces a memory access—or at least a cache access—to fetch the next instruction byte).

As a matter of fact, some people recommend two JMP $+2 instructions between I/O accesses to the same port, and three jumps between I/O accesses to the same port that go in opposite directions (OUT followed by IN or IN followed by OUT). This is clearly necessary when accessing certain motherboard chips, but I don't know how applicable it is when accessing VGAs, so make of it what you will. Input from knowledgeable readers is eagerly solicited. In the meantime, if you can use the BIOS to set the DAC, do so. In that case, you won't have to worry about the real and potential complications of setting the DAC directly.

AN EXAMPLE

I was going to finish this installment by covering reading the DAC, passing some examples of setting the palette RAM and DAC in both 16- and 256-color modes by you, and getting into color paging and cycling code, but I've again run out of space just as I was building up a head of steam.

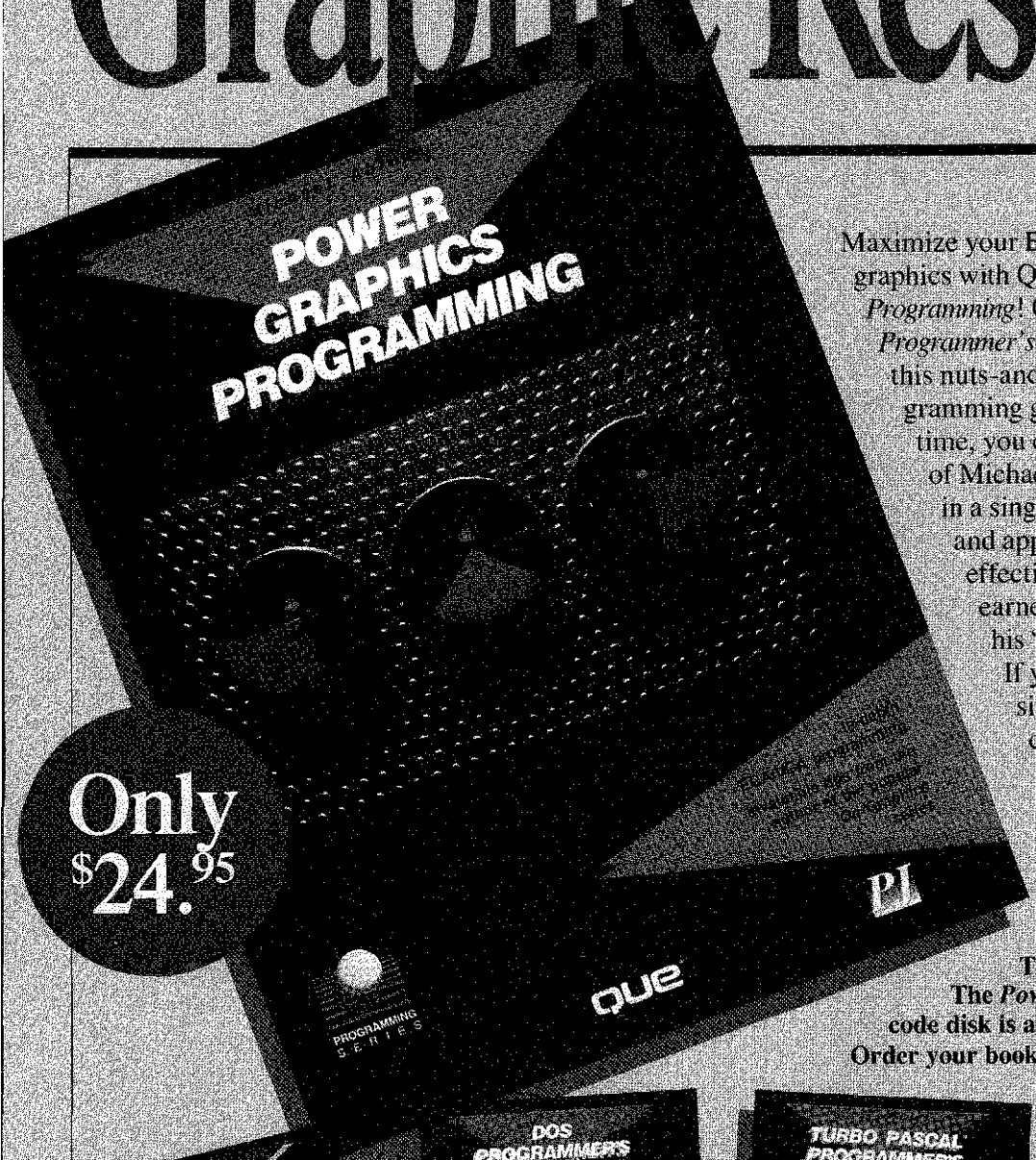
Quickly, then, Listing 1 is a simple example of setting the DAC that gives you a taste of the spectacular effects that color translation makes possible. There's nothing particularly complex about Listing 1; it just selects 256-color mode, fills the screen with one-pixel-wide concentric diamonds drawn with sequential attributes, and sets the DAC



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The complete *Programmer's Journal* "On Graphics" series—assembled in one volume for the very first time!

Graphic Results!

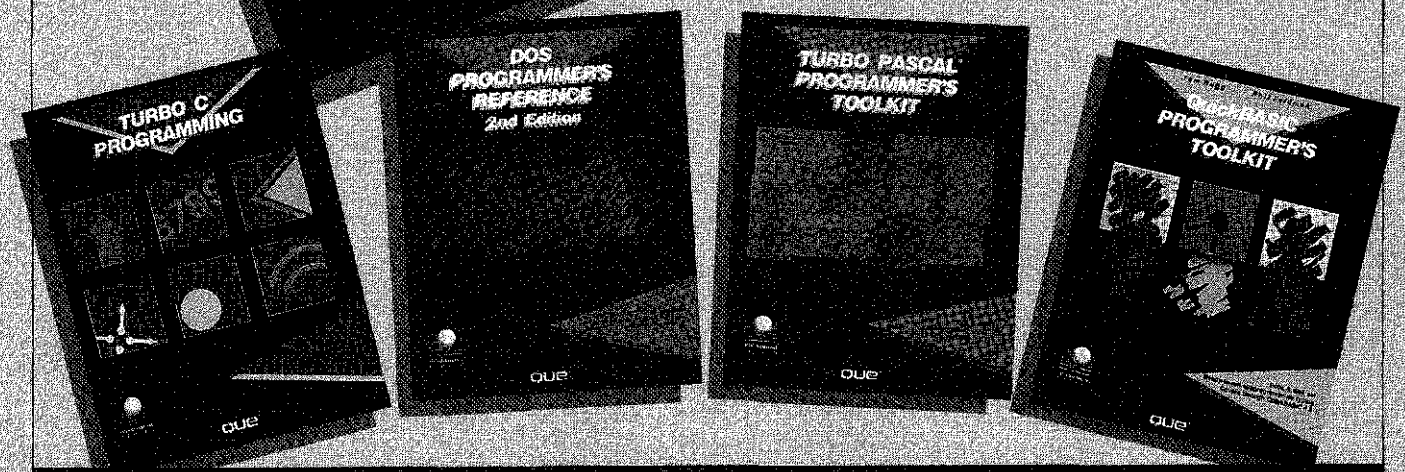


Only $24.95

Maximize your EGA/VGA programming graphics with Que's *Power Graphics Programming*! Que has teamed up with *Programmer's Journal* to bring you this nuts-and-bolts graphics programming guide. Now, for the first time, you can have the expertise of Michael Abrash consolidated in a single volume. His insights and approach to efficient, effective programming have earned him high acclaim for his "On Graphics" series. If you are seeking that elusive graphic edge, you can't be without *Power Graphics Programming*!

Order yours directly from *Programmer's Journal*

To order call 800-234-0386. The *Power Graphics Programming* code disk is also available for $14.95. Order your book and code disk today!



Look for these other practical Programming books from Que

Available in better bookstores and computer stores everywhere. Or call 800-428-5331, ext. PJ01.

que®

Circle 296 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Microsoft Windows Developers Manage Structures with Ease and Elegance

Ever notice how easy it is to miss a few loose ends during error recovery or termination, and wind up with outstanding memory allocations? The SPG **Structure Set Manager** (**SSM**) insures that there are no loose ends. With **SSM** you can easily identify all dynamic structures belonging to a window and release them when required. Each time you create a new dynamic structure, **SSM** allows you to specify the set(s) of which it is a member. To specify a set, you provide a name string to signify, for example, the window for which the structure was created. Using **SSM** functions you can control the creation and disposition of your own structures, as well as of SDK resources, with ease and elegance. With **SSM** you can:

- Create a set with a given name
- Add or remove a structure to or from a set
- Enumerate all members of a given set
- Ascertain all sets for which a structure is a member
- Obtain unions and intersections of sets
- Abolish a set

The basic package contains the **SSM** dynamic link library and user-level documentation, complete with instructions and examples. The examples show how use of this innovative tool can simplify your program logic and insure robust exception processing. The source package contains all items in the basic package, plus the source file for the the **SSM** library and auxiliary files needed to create the dynamic link library. A program logic manual is also provided.

Basic Package	**3 1/2 or 5 1/4 disk**	**$ 65.00**
Source Package	**3 1/2 or 5 1/4 disk**	**$175.00**

If you wish to place an order or desire further information, please write to the address below. If ordering, include check or money order and specify your MS Windows version number and preferred diskette size.

Software Projects Group
3451 Phoebe Court
West Covina, California 91792
(818) 913-7170

Circle **356** on Reader Service Card

to produce a smooth gradient of each of the three primary colors and of a mix of red and blue. Run the program; I suspect you'll be surprised at the stunning display this short program produces. Color translation is perhaps the easiest way to produce truly eye-catching effects on the PC.

Note the jagged lines at the corners of the screen when you run Listing 1. This shows how coarse the 320x200 resolution of mode 13h actually is. Now look at how smoothly the colors blend together in the rest of the screen. This is an excellent example of how careful color selection can boost perceived resolution, as for example when drawing antialiased lines (about which more another time).

Finally, note that the border of the screen turns green when Listing 1 is run. Listing 1 reprograms DAC register 0 to green, and the border attribute (in the Overscan register) happens to be 0, so the border comes out green even though we haven't touched the Overscan register. Normally, attribute 0 is black, causing the border to vanish, but the border is an 8-bit attribute that has to pass through the DAC just like any other pixel value, and it's just as subject to DAC color translation as are the pixels controlled by display memory. However, the border color is not affected by the palette RAM or by the Color Select register.

IN CONCLUSION

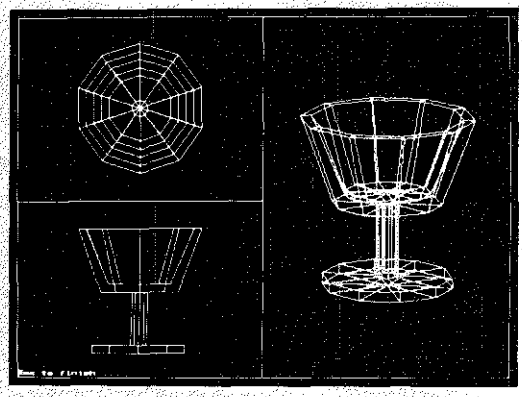
In this installment we traced the surprisingly complex path by which the VGA turns a pixel value into RGB analog signals headed for the monitor. In the next installment we'll look at some more code that plays with VGA color, we'll save and restore the palette RAM and DAC settings, and we'll observe color paging and cycling in action. After that, who knows? As always, your suggestions are more than welcome.

See you next time.

THE ULTIMATE CAD/CAM ENGINE

TurboGeometry Library 3.0!

Helps You build it right THE FIRST TIME!



Over 300 2D & 3D device independent geometric routines. Surfacing; Spherical & Solid Geometry; Tangents; Curves; Hidden Line; Polygons; Perspectives; 2D & 3D Clipping; Link List Library & more. In C or Pascal, w/source. 400 pg manual. $199.95, Foreign $225.00. Prices incl S&H. 30 day guarantee. VISA, MC, PO, Chk.

2116 E. Arapaho Rd.
Suite 487
Richardson, TX 75081
USA

Disk Software, Inc.

FAX or Phone
214-423-7288
Toll Free
1-800-635-7760

Circle **229** on Reader Service Card

Tired of the limitations of MS-DOS?
Thinking about switching to another OS?
Well DON'T, now there is:

PRO-CLI v4.1

CLI is a tiny, intelligent front & back end processor that enhances the MS-DOS system kernel. CLI provides many features of advanced OS's, like VMS & UNIX, to MS-DOS without the tremendous costs and additional training.

CLI-Types: ALIASs, LOGICALs, SYMBOLs, LEXICAL-FUNCTIONs and STREAMs are available to the command line, batch programs, & all existing application programs. CLI is 100% MS-DOS compatible, and NO changes are required in any software to reference any CLI-Types.

Some of the features offered by PRO-CLI include:

- batch programs with numeric & string processing
- facilities for source code and project management
- transparent task-to-task communications
- file & device management from the command line
- command line editing, history, and recall
- flexible system installations & re-configurations
- user configurable PRO-CLI resources

For the first time, designers, developers, and managers can coordinate software development & production efforts without the major restrictions of MS-DOS.

Call or Write for Additional Information!

Trademarks: VMS of D.E.C. MS-DOS of Microsoft UNIX of AT&T

Alphax Systems, Inc.
1005 Sussex Blvd.
Broomall, PA 19008
(215) 543-2658

Price: $ 149
S/H: $ 6

Major Credit Cards Accepted

Circle **345** on Reader Service Card

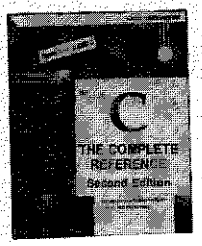
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE COMPUTER PROFESSIONALS' BOOK SOCIETY

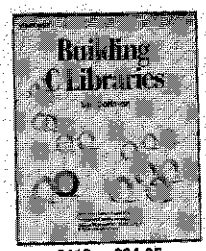
3 books for only $2.95

plus a 4th book of your choice FREE!

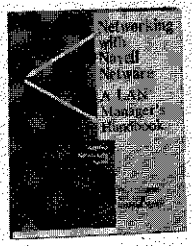
Values to $150.35



9309P $28.95
Counts as 2



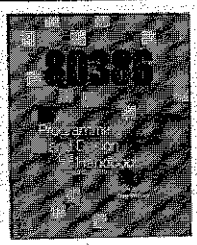
3418 $34.95
Counts as 2



3283 $28.95

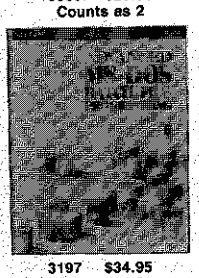
9291 $29.00
Counts as 2

15034P $24.95

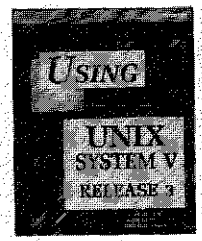


3237P $24.95

10015 $42.95
Counts as 2

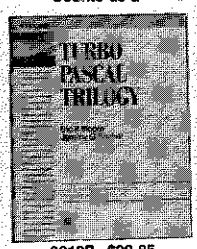


3197 $34.95
Counts as 2



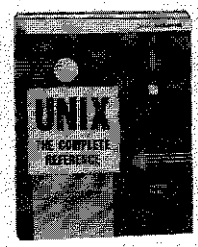
15016P $27.95

3049 $37.95
Counts as 2

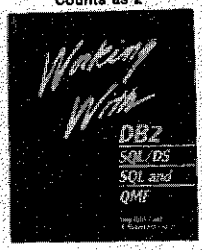


3310P $29.95
Counts as 2

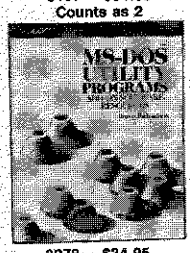
2917P $16.95



9258P $29.95

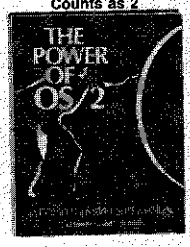


3508 $32.95

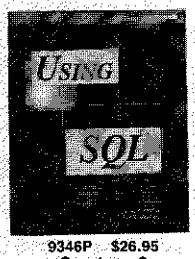


3278 $34.95
Counts as 2

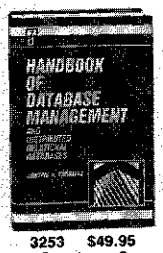
9808 $34.95



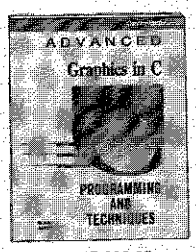
2993 $29.95



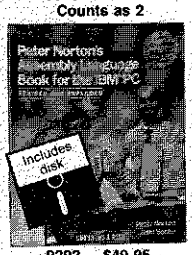
9346P $26.95
Counts as 2



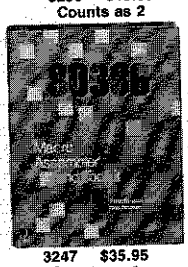
3253 $49.95
Counts as 2



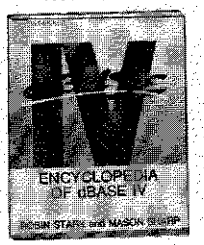
15001P $22.95



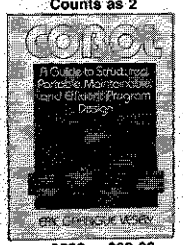
9292 $49.95
Counts as 2



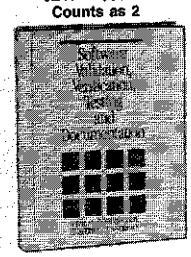
3247 $35.95
Counts as 2



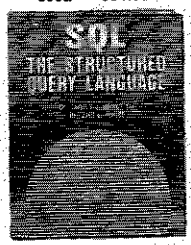
3332 $34.95



9323 $39.00
Counts as 2



8147 $49.50



3016P $17.95

How the Club Works

YOUR BENEFITS: You get 4 books for $2.95 plus shipping and handling when you join. You keep on saving with discounts of up to 50% off as a member.

YOUR PROFESSIONAL BOOKSTORE BY MAIL: Every 3-4 weeks, you will receive the Computer Professionals' Book Society News describing the Main Selection and Alternates, as well as bonus offers and special sales, with scores of titles to choose from.

AUTOMATIC ORDER: If you want the Main Selection, do nothing and it will be sent to you automatically. If you prefer another selection, or no selection at all, simply indicate your choice on the reply form provided. You will have at least 10 days to decide. As a member, you agree to purchase at least 3 books within the next 2 years and may resign at any time thereafter.

BONUS BOOKS: Starting immediately, you will be eligible for our *Bonus Book Plan*, with savings of up to 80% off publishers' prices.

IRONCLAD NO-RISK GUARANTEE: If not satisfied with your books, return them within 10 days without obligation!

EXCEPTIONAL QUALITY: All books are quality publishers' editions especially selected by our Editorial Board.

(Publishers' Prices Shown)

The easy, reliable way to satisfy your professional book needs.

Computer Professionals' Book Society
Blue Ridge Summit, PA 17294-0870

Please accept my membership in the Computer Professionals' Book Society and send the 4 volumes listed below, billing me $2.95. If not satisfied, I may return the books within ten days without obligation and have my membership cancelled. I agree to purchase at least 3 books at regular Club prices during the next 2 years and may resign any time thereafter. A shipping/handling charge and sales tax will be added to all orders.

			FREE BOOK*

Name _____

Name of Firm _____
(If you want Society Bulletins sent to your office)

Address _____

City _____

State _____ Zip _____ Phone _____

Valid for new members only. Foreign applicants will receive special ordering instructions. Canada *must* remit in U.S. currency. This order subject to acceptance by the Computer Professionals' Book Society. PJ990

Call 1-800-233-1128 to Join Today!

*Not valid for books counting as 2 selections.
All books are hardcover unless number is followed by a "P" for paperback.
©1990 Computer Professionals' Book Society, Blue Ridge Summit, PA 17294-0870

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Implementing a virtual data system and virtual arrays

A virtual memory system in C/C++

By Allen Holub

As computers get smaller and memory gets cheaper, a sort of memory chauvinism creeps into many a programmer's thinking: "*Real* machines have 32 bits of addressable memory—don't bother me with your whining about portability." This sort of thinking leads to Godzilla programs that gobble up megabytes of memory as they run amok through a system. To my thinking, it's inexcusable for a program not to function simply because there's not enough RAM. Programs should use all the memory available on the computer—not just core and LIM memory, but the disk as well.

This article looks at a small, disk-based virtual memory system that you can use to store large arrays. It's configured to support 16 megabytes of memory, but is easily reconfigured to support more. Since the disk can be a RAM disk, it also lets you use your extra RAM without having to worry about LIM paging, etc. The virtual-memory manager is written in C for maximum flexibility. I also incorporate the C functions into a C++ class that implements very large arrays of **int**s in a reasonably painless fashion. For example, you can declare a 10-megabyte array of **int**s with:

```
vint a( 1024L * 1024L * 10L );
```

and then access array elements using normal array notation:

```
a[10] = a[100000L] + a[i];
```

It's a simple matter to adapt these functions to arrays of other types as well.

All virtual-memory systems are based on a paging strategy. In a hardware system, for example, the virtual-memory manager detects a read from or write to an address for which there's no physical memory. The manager then fiddles with some bits so that the machine is actually accessing a small RAM buffer; the program is talking to real memory at a different address entirely. The data that ought to be in the virtual-memory region is then transferred from the disk to the internal buffer. As long as the virtual address falls within the buffer, the machine is just accessing RAM, so there's no performance degradation. When the address gets too large or small, however, the current buffer must be swapped back out to disk, and a different part of the virtual-memory file is read in to replace it. Of course, if the program has only read from virtual memory, the new buffer can be loaded with a single disk read without having to save the existing buffer back to disk first.

VIRTUAL ARRAY TACTICS

The system presented here follows roughly the same scheme. When you open a virtual array, enough space is allocated from a large disk file to hold the entire array. At the same time, a structure is created in memory that contains, among other things, a small buffer (currently 256 bytes).

The disk file is broken up into several blocks, each of which represents one bufferful of data. When a specific array index is requested, the read function determines where on the disk (in which block) that array element is found. It then transfers the correct block into the local buffer and returns a pointer to the place in the buffer where the copy of the array element resides. The program can read the memory through this pointer, but since this isn't done via hardware, the program must call a function to write to the memory. A function call is required so the swapping function can know that the buffer has been modified—otherwise, it wouldn't know whether or not to update the disk block corresponding to the current buffer before overwriting that buffer. A disk transfer doesn't occur unless an array element that is not already in the buffer is accessed.

In addition to the actual virtual-array access routines, there is a set of memory-management routines. It's not feasible to reduce the size of the virtual-memory file, but there's no reason why the existing disk space can't be recycled. The

---

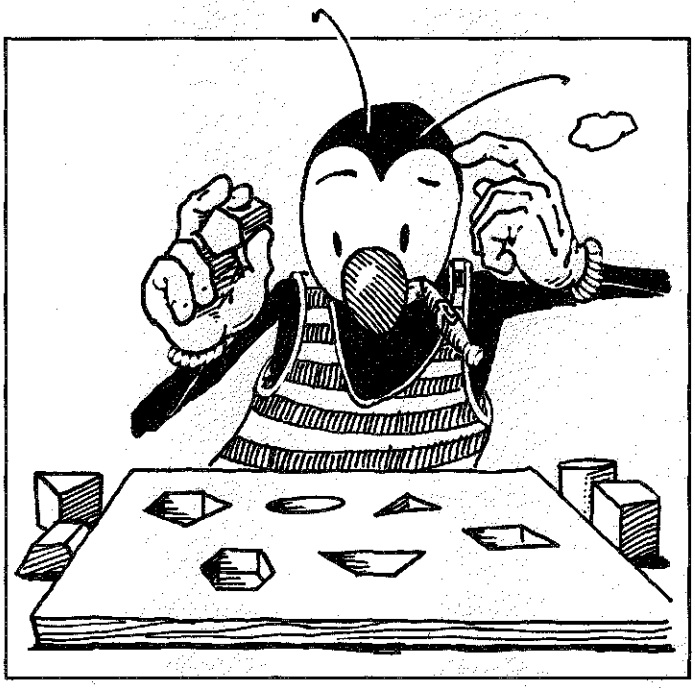
Allen Holub is a teacher, writer, consultant and composer. He teaches C and compiler design for the U.C. Berkeley Extension and has authored several books, most recently Compiler Design in C *from Prentice Hall. You can reach him electronically at holub@violet.berkeley.edu (...!ucbvax!violet!holub) via USENET or c/o Software Engineering Consultants, P.O.Box 5679, Berkeley, CA 94705 by US snail. His scotch is not Dewers.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

memory manager keeps track of available regions of the virtual-memory file and allocates from those, if possible.

Virtual arrays are implemented using a small set of functions. All these functions return some sort of error code if necessary, and the global variable *errno* is set to a value which reflects what went wrong. The first two are:

```
int  vopen ( void );
int  vclose( void );
```



These functions initialize and shut down the virtual-memory system. The function **vopen** creates the virtual-memory file *$$$$vmem.tmp*. If a TMP environment already exists, its contents are appended to the front of the file name. A slash is added to the end of the name in TMP if one isn't already there. The **vclose** function cleans up. It deletes the virtual-memory file and clears various internal data structures so that the system can be reopened later on with no ill effects. Both functions return true on success and false on failure, setting *errno*.

Once the system is initialized, a virtual array is allocated:

```
void *handle;
handle = vmalloc( long number_of_elements,
                  size_t element_size_in_bytes );
```

This function allocates the space for the file on the disk, creates the internal data structure needed to manipulate the memory, and returns a pointer to this internal data structure. The internal data structure is used much like a file handle: it must be passed to all the functions that manipulate this array. Every array has its own internal buffer to make transfers from one array to another a little more efficient. The function **vmalloc** returns NULL on failure and sets *errno* as appropriate. Note that the *number_of_elements* argument is a **long**, not **size_t** as is the case in **calloc**. Also unlike **calloc**, the allocated array is not initialized.

Two other functions manage memory:

```
int  vfree( void *handle );
long vele ( void *handle );
```

Both are passed handles returned from previous **vmalloc** calls. The function **vfree** frees up the disk space used by the array. (Subsequent attempts to access that array will generate an error code.) Function **vele** returns the number of elements in the array (an identical number to the first argument to **vmalloc**). Both routines return false on an error and set *errno*.

The final three functions are used to access array elements:

```
void *vread ( void *p, long index );
void *vwrite( void *p, long index, void *object );
void  vdirty( void *p )
```

The **vread** function gets the indicated array element into the local buffer and returns a pointer to that element in memory. The **vwrite** function copies the object pointed to by the last argument to the indicated array index and returns a pointer to the array element's current location in memory. Both functions return NULL (and set *errno*) on an error. As an example:

```
void *handle;
vopen();
...
/* allocate a 100,000-element
   array of longs */
if( !(handle = vmalloc( 100000L,
    sizeof(long) )))
    perror( "vmalloc()" );
...
/* add 5 to array[10] */
x = *(long *)vread( handle, 10L );
x += 5;
vwrite( handle, 10L, &x );
...
vfree( handle );
vclose();
```

I've left out most of the error checking in this example so that you can see what's going on.

The **vdirty** function can be used to speed up the writing process, since **vwrite** isn't particularly efficient. You can use the pointer returned from **vread** to modify the data, but you must tell the virtual-memory manager that the buffer has been modified or the modifications won't be stored back onto the disk when the next swap occurs. Do this with a **vdirty** call:

```
long *p;
p = (long *)vread( handle, 10L );
*p += 5;
vdirty( handle ); /* force a swap */
```

The next layer in the system uses C++ to make the virtual-memory interface a little easier to use. I've implemented virtual arrays of **int**s, though the class definitions are easily adapted to other types. To allocate a 10,000-element array:

```
vint a(10000);
```

Thereafter, you can access an array element in the usual way, assign to it, etc. I've implemented addition, subtraction, multiplication, division, and assignment operators. The +=, -=, etc. operators are not implemented, just the assignment operator. The ++ and -- are implemented but, as is required by a deficiency in C++, postfix and prefix are not distinguished. They are always treated as prefix expressions, even if the ++ or -- follows the operand.

ONE C IMPLEMENTATION

The low-level, C virtual-memory functions are implemented in Listing 1. The code compiles under both Microsoft C (5.1) and Zortech C++ (2.0), though Zortech kicks out hogsheads of garbage in the guise of "possible unintended assignment" warnings. You can turn these off with a *-w* on the command line, but you will lose all the warning messages.

The listing starts with several macro definitions and typedefs. The **D(x)** and **ND(x)** macros facilitate debugging. When DEBUG is **#defined**, the **D(x)** macro expands to its argument and the **ND(x)** macro expands to an empty string, discarding

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the argument. The behavior is reversed when DEBUG hasn't been defined. For example:

```
D( printf("Printed only if DEBUG #defined" );)
ND(printf("Printed only if DEBUG isn't #defined");)
```

These macros are preferable to putting **#ifdef DEBUG ... #endif** statements all over the place. They take up fewer lines, and they don't mess up your indenting.

The first section of interesting code is **vopen** in Listing 1. The function **vopen** starts out by assembling the file name from the TMP environment and the **TNAME** macro. It then opens the file. The only strangeness here is the **creat** call, which is needed to get the code to port to the Zortech compiler. Zortech doesn't support an **O_TRUNC** mode to **open**, but **creat** will truncate the file to zero length when it creates it. Rather than paying attention to the permission mask as it should, **creat** always opens the file for writing. Since I need the file open for both reading and writing, I have to close it and then reopen it with an **open** call.

VIRTUAL MEMORY ALLOCATION

The next set of routines are the memory-management functions **vmalloc** and **vfree**. BLK_SIZE sets the size in bytes of both the internal buffer and the block size on the disk. Various quantities, such as the file size, are stored internally in terms of blocks, not bytes. The offset from the base address of the file to the beginning of a particular array is also stored in terms of blocks. These numbers are all **unsigned ints**, so if you have a 16-bit **int**, the largest block number in the file will be 65,535. (Block 0 is the first one in the file.) The largest file (and object) is then 65,536*BLK_SIZE bytes. Since BLK_SIZE is set to 256 here, the file size is limited to 16,384 Kbytes (16 megabytes). Increasing the block size gives you more room on the disk, but every object will require a larger internal buffer, so more memory will be used. On the other hand, the larger the internal buffer, the less often the virtual-memory manager will have to go to the disk to fetch new data, so array manipulation will be more efficient when you're doing things like sequential reads and writes. Tune the block size to your own application.

Two data structures are used to manage the virtual memory: **hdr** and **vmem**. The **hdr** contains information that is needed by both the memory-allocation system (**vmalloc** and **vfree**) and the virtual-array functions (**vread** and **vwrite**). The **vmem** structure has the information that is used only by the virtual-array functions.

I'll demonstrate how this works by describing the memory-allocation process. The allocator keeps a "free list" that tells it which areas of the disk are available.This is a circular-linked list of **hdr** structures, declared as follows:

```
typedef struct hdr {
    unsigned    nblocks;
    unsigned    offset;
    union {
        struct hdr  *h;
        struct vmem *v;
    } next;
} hdr;
```

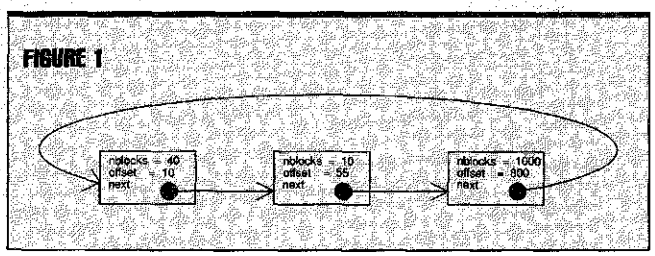
The **nblocks** field identifies the number of BLK_SIZE-sized blocks that are available. The **offset** field is the offset, in blocks, from the base of the file to the first of the blocks. The **hdr.next.h** field points at the next entry of the linked list. (It's

Listing 1

```
#include <stdio.h>
#include <io.h>
#include <fcntl.h>      /* for Microsoft-compatible O_RDWR */
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>

 /* VIRT.C.   This file contains C workhorse functions that manipulate
  * disk-based virtual memory. */

#ifdef DEBUG
#define D(x) x
#define ND(x)
#else
#define D(x)
#define ND(x) x
#endif
#define READ  1                 /* first argument to block_io()      */
#define WRITE 0                 /* "                                 */
#define TNAME    "$$$$vmem.tmp" /* name used for virtual-memory      */
                                /* file. TMP environment prefix      */
#define BLK_SIZE  256           /* block size (bytes)                */
#define SIGNATURE 0x5a5a        /* Identifies vmem's as valid.       */
                                /* Stored in 15-bit unsigned.        */
static char     Vname[128];     /* path name to vmem file            */
static unsigned Filesize = 0;   /* size of virt mem file (blocks)    */
static int      Vfd      = -1;  /* File handle for Vname             */
typedef struct hdr {
    unsigned    nblocks;        /* Size of array in blocks.          */
    unsigned    offset;         /* Offset (blocks) to block start.   */
    union { struct hdr  *h;     /* Next element of freelist.         */
        struct vmem *v;         /* Additional info if in use.        */
    } next;
} hdr;
typedef struct vmem {
    unsigned signature:15;      /* 0x5a5a if structure valid         */
    unsigned dirty    :1;       /* true if buf modified since read   */
    unsigned ele_size;          /* size of one element               */
    unsigned ele_per_blk;       /* number of elements in a block     */
    long     num_ele;           /* number of elements in array       */
    unsigned cblock;            /* block currently in buffer         */
    char     *buf;              /* Swap buffer                       */
} vmem;
#define PUBLIC                  /* for documentation                 */
#define PRIVATE static
PRIVATE hdr *Freelist = NULL;   /* Pointer to free list--circular    */
                                /* linked list, initially empty      */

extern  int vopen       ( void );
extern  int vclose      ( void );
extern  hdr *vmalloc    (long num_ele,unsigned int ele_size);
extern  int vfree       (hdr *p);
extern  void *vread     (hdr *p,long index);
extern  void *vwrite    (hdr *p,long index,char *this);
extern  long vele       (hdr *p);
static  hdr *new_hdr     (unsigned nblocks, unsigned offset);
static  unsigned enlarge (unsigned blocks_reqd);
static  hdr *add_vmem    (long num_ele, int ele_size, hdr *header);
static  int load_block   (hdr *p, unsigned req_block);
static  int block_io     (int doread, unsigned block, char *buf);
/*-------------------------------------------------------------------------*/
PUBLIC int vopen() {
/* Open the virtual-memory system: Create the virtual-memory file and open
 * it for read and write in unbuffered mode. Return true on success, false
 * on failure (errno is set to an appropriate value in this case). The
 * file is called "$$$$vmem.tmp", and it will be put into the directory
 * specified in the TMP environment if this environment is present. E2BIG
 * is used if the path in the TMP environment is too long. */
    char *env;
    int  len;
    if ( !(env = getenv( "TMP" )) )     /* Assemble file name */
        strcpy( Vname, TNAME );         /* No TMP environment */
    else {
        /* If TMP environment exists, use it as a filename prefix
         * adding a trailing / if necessary. */
        if ((len = strlen( env )) > sizeof(Vname) - 16) {
            errno = E2BIG;
            return 0;
        }
        sprintf( Vname,
            ( len>=2 && env[len-2]!='/' && env[len-2]!='\\' ) ?
                            "%s/%s" : "%s%s", env, TNAME );
    }
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



put into a union to avoid having to cast it all the time.) A small free list is pictured in Figure 1.

The strategy used here is essentially the same one that's used by **malloc** and **free** in Kernighan and Ritchie's *C Programming Language*. The **hdr** structures each represent an available group of contiguous blocks in the disk file. They are arranged in ascending order of offset, with the *freelist* pointer referencing the node to the left of the most recently accessed node.

The free list starts out empty (*freelist* is NULL). The **hdr** structure for each group of blocks is created by **vmalloc** when the virtual array is created. A pointer to this **hdr** is returned from **vmalloc** as a handle, and this pointer is eventually passed to **vfree** when the space is deallocated. **vfree** then puts the node into the correct part of the free list. If the freed space is adjacent to any space that's already in the free list, it's coalesced with the existing node rather than being inserted into the list. For example, 10-block group in the middle of the free list in Figure 1 extends from offset 55 to offset 64 on the disk. A block starting at offset 65 can be added to the list by adding the new block's size to the existing block's size. If the new group occupied 735 blocks, it would extend all the way up to the group represented by the **hdr** at the far right of the list. In this case the sizes of the new, middle, and right **hdr**s could all be added together and put into the middle **hdr**. The right one could then be removed from the list.

Most of the complexity in **vmalloc** has to do with searching the free list. The list is searched with two pointers, which chase each other around the list: *cur* points at the current node and *prev* points at the one to the immediate left of *cur*. The first **if** statement detects whether a block is physically the last block in the file. The next test detects whether a block is big enough. If the block is bigger than necessary, it is made smaller by decrementing the **nblocks** field. A new **hdr** is created and initialized to represent the space we just opened up.

Note the **goto** *end* branch in **vmalloc**, which is working as a **return** statement here. In general, I feel that having multiple **goto** branches to a single return statement makes more sense than multiple return statements. The current routine is a good example why. I wanted a debugging diagnostic at the end of the routine that told me what **vmalloc** was returning and what

SVS
WINDOWING SCREEN MANAGER
Finally, the windowing screen manager you've been waiting for!
Tired of the frustration of forcing square pegs into round holes? So are we! That's why we created the SVS screen manager, the flexible manager that works for you. It's easy to use! It's ideal for large scale team development of user interactive applications. It's NOT another resource pig! With SVS you can prototype a complete system in minutes. You can even have numerous virtual screens open in multiple windows, but only if you want them. SVS doesn't force anything.
Professional Development Version 2.0 $320.00
Screen Management by Example
Structured Windowing Language
System Prototyping Tools
Screen Porting Facilities
Network & Mouse Support
Builds Menu Shells for DOS
Graphics Library for "C"
Sight Licensing and Demo Available
Supports: DOS, C, Pascal
Inspired Innovations
4001 South Decatur Boulevard, Suite 332
Las Vegas, NV 89103
PHONE 1-800-752-8874
Circle 358 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

it did to the free list. The **goto** lets me have one such diagnostic, instead of having to sprinkle diagnostics all over the place.

If the block in the free list is exactly the right size, the **hdr** representing the block is actually removed from the free list, and a pointer to the existing **hdr** is returned. In both situations, the *next* pointer in the **hdr** is recycled at this point. The **add_vmem** function creates and initializes a **vmem** structure for the current object and then makes the *next* field in the **hdr** point at the new **vmem**. This is why the *next* field is a union—so you don't have to cast it to **(vmem*)** every time you use it.

If no block is big enough, the code then looks to see if a group of blocks that represent space that is physically at the end of the file can be found. If so, the file is enlarged and the block count of this last node is adjusted. The code will then loop back up to the free list searching code, which will now find the block we just modified. If no block was big enough and no block could be expanded to make it big enough, then an entire new block is added. In this case, a new **hdr** is created, and a pointer to the new **hdr** is returned.

The job of the **vfree** function is somewhat easier than **vmalloc**'s. The only complication is the search for the place where the new node belongs (in the "for" loop). The node can be at the "corner" of the list (where the list wraps from the group with the largest offset to the group with the smallest) and the extra complexity takes care of this special case. The pointer *cur* will indicate the node that will be to the left of the new node.

The following "if" statement takes care of the degraded case of a free list with only one **hdr**, while the code at the bottom handles the normal case. You should take a moment to study the strategy here, taken almost verbatim from Kernighan and Ritchie. Each clause handles one side of the node being added. The first clause takes care of the *next* pointer on the node being added. It either coalesces the new node with the existing one, or makes the *next* pointer point at the existing one, as required. The same thing is done with the predecessor node.

VIRTUAL MEMORY FUNCTIONS

The remainder of Listing 1 handles the virtual-memory access. It's much easier to understand than memory-management. A read is done by **vread**. It figures the block number of the disk area that holds the required array element, and then calls **load_block** to get it. The next step determines where in memory the desired array element was actually loaded and returns a pointer to that space.

The **load_block** function first checks to see if the desired block is already in memory. If so, it just returns. If not, it looks to see if the block has been modified since it was loaded. The function does this by examining the current object's **dirty** bit, which is set true by **vwrite** every time the buffer is modified. If the buffer has been modified, **load_block** writes the buffer out to the disk before doing the load.

The **vwrite** function does the writing. It calls **vread** to find out where to write, and then transfers the object into memory one byte at a time. It then sets the **dirty** bit in the **vmem** structure to tell the swapping code that the current buffer needs to be written to disk before loading another block on top of it. This is one place where performance can be improved considerably. While **vwrite** is safe, it is not particularly fast.

A speedier approach uses **vread** to load the array element, modifies that cell directly by using the pointer that was returned from **vread**, and then sets the **dirty** bit explicitly to tell the memory manager that a swap-out will be required before loading the next buffer. The **vdirty** subroutine is defined for this purpose, setting the **dirty** bit for the current buffer. Be wary if you use it: it's a serious problem if you happen to forget to set the **dirty** bit after you write to an array element.

Listings 2 and 3 provide function prototypes for the virtual-memory functions. Listing 2 (VIRT.H) provides ANSI-C-style prototypes. Listing 3 (VIRT.HPP) makes these prototypes suitable for inclusion in a C++ program by surrounding them with an **extern "C"** directive. (The C++ compiler needs to know if something is a C function so that it doesn't mangle the function name when it's called.)

A C++ IMPLEMENTATION

The C++ layer in Listing 4 (VINT.CPP) uses the virtual-memory functions to implement an array-of-int type. First, note that this is a header file, meant to be **#included** in a C++ source file. The difficulty here is that a C++ class definition contains both definitions and declarations. That is, the actual class definition must be passed around in an include file just as a C typedef would.

However, the methods (the subroutines that manipulate the class) must be compiled once and put into a library, from which the linker calls them into the program. For maintenance reasons, you'd like both the class definition and the methods to be in the same file. This is accomplished by surrounding everything that generates code (i.e., the methods) with a **#ifdef COMPILE_METHODS** and **#endif**. You create a file called VMETHOD.CPP that looks like this:

```
#define COMPILE_METHODS
#include "vint.cpp"
```

You can then compile VMETHOD.CPP in the normal way and put the resulting object module into a library. Everywhere else, just include VINT.CPP without defining COMPILE_METHODS.

The next problem has to do with implementing the bracket operator in a reasonable way. A normal C++ array class would use a reference to generate an lvalue for an array element. For example, in this class,

```
class array
{
    int array[100];
    ...
public:
    int &operator[ ]( int index )
    {
        return array[index];
    }
    ...
}
```

the [ ] operator returns a reference to the correct array element. A program could allocate and access an object as in:

```
array x;        /* 100-element array */
x[10] = -1;     /* put -1 into x[10] */
```

The **x[10]** would call the **operator[ ]**method, which returns a reference to the correct array element, which is then modified via the reference. Unfortunately, this strategy can't be used in

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    /* Open file, truncating to zero length if it exists. We use
     * creat rather than open to avoid portability problems between
     * Microsoft and Zortech. The file is opened only for writing,
     * so the file is closed and reopened for reading and writing. */
    if ( (Vfd = creat(Vname, S_IREAD | S_IWRITE)) == -1 )
        return 0;
    close( Vfd );
    if ( (Vfd = open(Vname, O_RDWR | O_BINARY )) == -1 )
        return 0;
    return 1;
}
/*-------------------------------------------------------------------------*/
PUBLIC int vclose() {
/* Close the virtual-memory system and delete vmem file. Return 1 on
 * success, 0 on failure (and errno is set to something reasonable) EBADF
 *  is used if no previous vopen() has been issued. */
    hdr *p, *newp;

    if ( Vfd == -1 ) {                  /* the file isn't open */
        errno = EBADF;
        return 0;
    }
    else {
        if ( close( Vfd ) == -1 )       /* close & delete file */
            return 0;
        Vfd = -1;

        ND( if ( unlink( Vname ) == -1 )  )
        ND(     return 0;                 )

        if ( Freelist ) {               /* Empty the free list */
            p = Freelist;
            do {
                newp = p->next.h;
                free( p );
            } while ( (p = newp) != Freelist );
            Freelist = NULL;
        }
    }
    return 1;
}
/*-------------------------------------------------------------------------*/
PUBLIC hdr *vmalloc( long num_ele, size_t ele_size ) {
/* Allocate a virtual array, having num_ele, each of ele_size bytes.
 * Return a pointer to use in subsequent operations. The size of the
 * internal buffer used to access elements is controlled by vopen().
 * Return NULL if no virtual memory is available or there is a seek error.
 * Errno will have an appropriate value in it in this case. */

    hdr  *prev, *cur, *new, *eof_block ;
    unsigned ele_per_block;     /* elements per block       */
    unsigned blocks_reqd;       /* number of blocks required */

    D( printf("Entering vmalloc(%ld,%u). ", num_ele, ele_size); )
    D( pfreelist();                                             )

    ele_per_block = BLK_SIZE / ele_size ;
    blocks_reqd = num_ele/ele_per_block + (num_ele % ele_per_block != 0);

    if ( Freelist ) {    /* If there is a freelist */
        prev      = Freelist;
        cur       = Freelist->next.h;
        eof_block = NULL;

        while ( 1 ) {
            do {
                if ( cur->offset + cur->nblocks == Filesize )
                    eof_block = cur;    /* last block in file */

                if ( cur->nblocks >= blocks_reqd ) {
                    Freelist = prev;                    /* it's big enough */
                    if ( cur->nblocks > blocks_reqd ) {
                        cur->nblocks -= blocks_reqd;
                        if ( new = new_hdr( blocks_reqd,
                                    cur->offset + cur->nblocks) )
                            new = add_vmem( num_ele, ele_size, new );
                        goto end;
                    }
                    else {                      /* block is exact size */
                        if ( cur->next.h == cur ) /* no successor */
                            Freelist = NULL;
                        else
                            prev->next.h = cur->next.h;

                        new = add_vmem( num_ele, ele_size, cur );
```

continued

INTEL DEVELOPMENT TOOLS

For the ultimate success with your microcomputer development project Intel offers you the total solution. From emulators to compilers to performance analysis and simulation, Intel can supply you with the most up-to-date and powerful development tools available today.

We now offer you an easy method of ordering. Simply call our toll-free number at 1-800-874-6835 to request a complimentary copy of our Intel Development Tools Catalog listing all our offerings. Or drop us a line at: Intel Corporation, Catalog Sales, 5200 NE Elam Young Pkwy, JF1-15, Hillsboro, OR 97124. After you have had a chance to look over our products, give us a call. We can answer any questions you may have about Intel Development Tools and we'll be happy to take your order over the phone. No waiting. It's as simple as that. So call us today. We'll be waiting to hear from you.

CALL US TODAY 1-800-874-6835

Circle 50 on Reader Service Card

APPLIED INTUITION

ANNOUNCES

WINDOWS ++ VER 2.0

The Complete C++ User Interface Facility.

WINDOWING
- Virtual Screens
- Scroll & Move
- Dynamic Resizing
- Sub "NESTED" Windows
- Menus Bound to Windows
- Write Windows to Disk
- Write Virtual Screens to Disk

MENUS
- Pop Up
- Pull Down
- Spread Sheet
- Menu Compiler
- Context Sensitive Help

DATA ENTRY
- Form Design
- Date & Time
- Social Security #
- Telephone Numbers
- Currency
- Scientific Notation
- Complex Numbers
- Text with Editing
- Contex Sensitive Help
- Automatic Validation

SYSTEM SUPPORT
- PC/MSDOS
- VGA 29/43/50 Line Modes
- EGA 43 Line Mode
- CGA, MDA
- Unix, Xenix
- Curses, Terminfo, Termcap
- Mice
- C++ ver 1.2 or Greater

APPLIED INTUITION

5485 Beltline Rd. Ste. 225
Dallas, Texas 75240
CALL (214) 458-7134

Circle 41 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                    goto end;
                }
            }
            prev = cur;         /* go to next freelist element */
            cur  = cur->next.h;

        } while ( prev != Freelist );
        /* No block in the freelist was large enough. If a block
         * in the free list was at the end of the file, expand it. */
        if ( !eof_block )
            break;
        else if ( !enlarge(blocks_reqd - eof_block->nblocks) ) {
            new = NULL;
            goto end;
        }
        else
            eof_block->nblocks = blocks_reqd;
            /* Loop back up. The do/while will catch the
             * newly-expanded object. */
    }               /* end   while ( 1 )     */
}                   /* end   if ( Freelist ) */

/* Either there was no freelist, or no block was big enough in the
 * freelist and couldn't expand the last block. Make a new block. */
if ( !(new = new_hdr(blocks_reqd, Filesize)) )
    goto end;

if ( !enlarge( blocks_reqd ) )  {    /* changes Filesize */
    free( new );
    new = NULL;
}

new = add_vmem( num_ele, ele_size, new );
end:
D( printf("Leaving vmalloc(%ld,%u). Return:\n", num_ele, ele_size); )
D( pheader(new);    )
D( pfreelist();     )
D( printf("----------------------------------------\n"); )
return new;
}
/*---------------------------------------------------------------------*/
PRIVATE hdr *new_hdr( unsigned nblocks, unsigned offset ) {
    hdr *new;

    if ( !(new = (hdr *) malloc(sizeof(hdr))) ) {
        errno = ENOMEM;
        return NULL;
    }
    new->nblocks = nblocks;
    new->offset  = offset ;
    return new;
}
/*---------------------------------------------------------------------*/
PRIVATE unsigned enlarge( unsigned blocks_reqd ) {
/* Make the file blocks_reqd bytes larger by seeking that many bytes past
 * end of file. Adjust the global Filesize to indicate the real file size.
 * Return new filesize (in blocks) or 0 if the file couldn't be expanded.
 * Note that the second argument to lseek is an offset. Consequently, it's
 * one less than the count, thus the -1 in the assignment to position. The
 * 1 must be added back in the return statement, however, to make the count
 * work. The write() call both physically expands the file and advances the
 * file pointer one notch. This isn't necessary under UNIX, but MS-DOS
 * needs it. */
    unsigned size_in_blocks;
    long real_size;
    long position ;

    position = (long)blocks_reqd * BLK_SIZE - 1;

    if ( (real_size = lseek(Vfd, position, SEEK_END)) == -1) {
        errno = ENOMEM;
        return 0L;
    }

    if ( write( Vfd, "", 1 ) == -1 )
        return 0L;

    ++real_size;
    return( (real_size/BLK_SIZE > (unsigned)~0) ? 0 :
                (Filesize = (unsigned)(real_size / BLK_SIZE)) );
}
/*---------------------------------------------------------------------*/
PRIVATE hdr *add_vmem( long num_ele, int ele_size, hdr *header ) {
/* Create a new vmem structure and attach it to the header. Return header
 * argument or NULL if there was a problem. */
    vmem *p;

    if ( !(p = (vmem *)malloc( sizeof(vmem))) ) {        /* get the vmem */
        errno = ENOMEM;
        return NULL;
    }
    else if ( !(p->buf = (char *)malloc(BLK_SIZE)) ) {   /* & swap buf. */
        errno = ENOMEM;
        return NULL;
    }
    else {
        header->next.v = p;
        p->signature   = SIGNATURE;
        p->dirty       = 0;
        p->ele_size    = ele_size;
        p->ele_per_blk = BLK_SIZE / ele_size;
        p->num_ele     = num_ele;
        p->cblock      = 0 ;
        return header;
    }
}
/*---------------------------------------------------------------------*/
PUBLIC int vfree( hdr *p ) {
/* Free virtual array pointed to by p. Return true on success, false if the
 * input pointer was invalid (if the first element of the structure wasn't
 * the SIGNATURE). Errno is set to EBADF in this case. */
    int rval = 1;
    hdr *cur, *h;
    vmem *v = p->next.v;

    D( printf("Entering vfree(0x%x). ", p ); )
    D( pfreelist();                          )

    if ( v->signature == SIGNATURE )
        v->signature = 0;
    else {
        errno = EBADF;
        rval  = 0;
        goto end;
    }

    free( v->buf );     /* discard the swap buffer    */
    free( v );          /* discard the vmem component */

    if ( !Freelist ) {   /* Free list is empty. Create a new one. */
        Freelist = p->next.h = p;
        goto end;                       /* return default value */
    }

    /* Find out where the newly freed node goes: The list is a circular
     * linked list arranged in ascending order of offset. Since it's
     * circular, there's one place where cur->offset > cur->next.h->offset
     * (see 1, below). We have to test for the normal situation where the
     * new node comes between the current node and its successor. The rest
     * of the following if statement tests for the corner situations when
     * the new node's offset is either larger (at 2) or smaller (at 3)
     * than anything currently in the list. */
    for ( cur = Freelist ; 1 ; cur = cur->next.h ) {
       if (  ( cur->offset < p->offset &&
               p->offset < cur->next.h->offset
             ) ||
             ( cur->offset >= cur->next.h->offset &&   /* 1 */
                 ( cur->offset < p->offset ||           /*  2 */
                   p->offset < cur->next.h->offset     /* 3 */
                 )
             )
          )
             break;
    }
    /* Now link the node into the list, coalescing if necessary. First
     * handle degraded case of a free list with only one element. */
    if ( cur == cur->next.h ) {    /* only block in chain */
        if ( p->offset + p->nblocks == cur->offset ) {
            cur->offset   = p->offset;
            cur->nblocks += p->nblocks;
            free( p );
            goto end;
        }
        else if ( cur->offset + cur->nblocks == p->offset ) {
            cur->nblocks += p->nblocks;
            free( p );
            goto end;
        }
    }
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the current situation because the array element that's in memory (the one that we could return a reference to) is not the real array element. The real one is on the disk. A program that modified the memory version directly wouldn't modify the **dirty** bit for the buffer, so the buffer wouldn't be swapped out to disk the next time a block was loaded. The bracket-processing code can't set the dirty bit, or it would be set on a read as well as a write, which would force a swap-out every time a new buffer was loaded even if the current buffer hadn't been changed.

RETHINKING THE BRACKET OPERATOR

My solution to this problem is to rethink the way a bracket operator works. A bracket, here, is a *selector* operator. It sets things up so that the *next* operation can access the array element selected by the previous bracket. If the *next* operation modifies the array element, that operation will set the **dirty** bit. For example, a statement like **x[10]=1** works in this way:

(1) Process the **[10]**, do a **vread** on the specified array element, and remember the returned pointer in a local variable inside the class definition. The [ ] operator takes a reference to a **vint** as an operand, and it returns the same reference.

(2) Process the assignment operator. This operator takes a left operand of type reference-to-**vint** and an **int** right operand. It modifies the previously-selected array element by using the pointer returned from the **vread** call in the bracket-processing code. It then sets the **dirty** bit with a **vdirty** call. Overloads of nonassignment operators (such as +) just use the pointer initialized in the previous bracket processing. Since nothing has been modified, these methods don't set the **dirty** bit.

Let's look at how all this works. The **vint** class is defined in Listing 4. The data component of the class contains several things, the first of which is the **static int** *nobj*. Since this variable is static, only one instance (which is shared by all objects) is created. This variable holds a count of the number of objects currently in existence and is incremented by the constructor each time it creates an object. If this is the first object, then **vopen** is called to initialize the virtual-memory system. Conversely, *nobj* is decremented by the destructor and **vclose** is called if *nobj* has been set to zero. This way, the programmer using the **vint** class doesn't have to worry about forgetting to initialize or close the virtual-memory system before or after using a **vint** object.

The remaining data elements of the **vint** class are unique to each object: **hdr** is the pointer returned from **vmalloc** when the array is allocated, **cur_ele** is a pointer to the most recently selected array element, and **cur_index** is that element's index.

The rest of the class definition takes care of the methods. A few of these need to be looked at closely. First, a copy constructor is supplied, but copying one virtual array to another is an expensive operation. The existing destination array must be discarded, a new one created, and it must be initialized one element at a time. Use this constructor with care.

A **vint**-to-**int** conversion is also provided. This conversion is needed for expressions like **cout << x[10]**, in which no specific operation follows the element selection. The conversion operator causes problems with other operator overloads, however. There are three possible combinations of operand types for each operator:

New and Improved

TRANSLATOR 3.0
(Pascal to C)

- Now, even faster and more efficient.
- Translates Jensen and Wirth Pascal to ANSI C. Use with your Turbo Pascal.
- Provides interactive display while the translation proceeds.
- Includes the fastest search routines to provide super fast translations.
- Still many examples, explaining the fine points, in the User's Manual.

Still only
$99.95
(or upgrade from 2.X for $49.95)

MILTON BROWN SOFTWARE
935 Chester Avenue
San Marino, CA 91108
(818) 796-5527

Circle **240** on Reader Service Card

Program FASTER with ...

CC-RIDER

or the new **C++ RIDER**

POP-UP SOURCE BROWSERS FOR ANY EDITOR !

CROSS-REF Editing!

Instantly pop-up your symbol definitions with a *hypertext hot-key* inside any editor! The *Standard Edition* has Source View, Edit and Paste commands. And the powerful *Professional Edition* even lets you walk through all *uses* of a symbol in your code as you edit ! C++ RIDER fully supports C++ v2.1, including classes, constructors, overloading, etc.

Source Analysis!

Build a database of symbol information with our powerful ***source code analyzer*** which provides a wealth of useful output, like fully commented static and global function prototype #include files and a detailed log file showing WHERE AND HOW all your symbols are used. And it now uses expanded memory for ***lightning fast*** source code scanning!

QuickHelp, too!

CCSYM can also create a ***help database for your program*** which is compatible with Microsoft's help system, *QuickHelp*. All symbols in your application are accessible from QuickHelp's menus, with ***documentation extracted automatically from your original source code.*** Works with ANSI/Microsoft C, plus AT&T/Zortech/ Borland C++.

For DOS and OS/2
$89 / 179 / 279

Western Wares
(303) 327-4898
Box C Norwood, CO 81423

FREE DEMO DISK

Circle **100** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    /* Now handle normal case. Link or coalesce to the right first,
     * then to the left. */
    if ( p->offset + p->nblocks == cur->next.h->offset ) {
        /* The blocks are adjacent. Coalesce and free the extra header. */
        p->nblocks += cur->next.h->nblocks;
        p->next.h   = cur->next.h->next.h;
        free( cur->next.h );
    }
    else
        p->next.h = cur->next.h;
    if ( cur->offset + cur->nblocks == p->offset ) {
        cur->nblocks += p->nblocks;      /* Adjacent */
        cur->next.h   = p->next.h;
        free( p );
    }
    else
        cur->next.h = p;

end:
    D( printf("Leaving vfree(0x%x). Returning %d. ", p, rval ); )
    D( pfreelist();                                             )
    D( printf("-----------------------------------------\n"); )
    return rval;
}
/*-------------------------------------------------------------------------*/
PUBLIC void *vread ( hdr *p, long index ) {
/* Returns pointer to array element at indicated offset from the virtual
 * array pointed to by p. Returns NULL if offset is out of bounds or the
 * block couldn't be loaded (errno is set too). */
    unsigned block_num;       /* block in which index is found */
    vmem     *v = p->next.v;  /* vmem component of *p          */

    if ( index < 0  ||  index > v->num_ele ) {
        errno = ERANGE;       /* index is out of range */
        return 0;
    }
    if ( v->signature != SIGNATURE || Vfd == -1 ) {
        errno = EBADF ;       /* signature doesn't match */
        return 0;
    }

    block_num = index / v->ele_per_blk ;
    if ( !load_block( p, block_num ) )
        return NULL;
    /* get the pointer:
     * (1) adjust index relative to the start of the current block
     * (2) Multiply by the element size to get a byte offset.
     * (3) Add to base address of buffer to get pointer. */
    index -= block_num * v->ele_per_blk;        /* (1) */
    index *= v->ele_size;                       /* (2) */
    return( v->buf + index );                   /* (3) */
}
/*-------------------------------------------------------------------------*/
PUBLIC void *vwrite( hdr *p, long index, char *this ) {
/* Transfer *this to the virtual array at the indicated index. Return a
 * pointer to the actual memory location for *this or NULL on an i/o
 * failure or a bad "p" argument. */

    int  i;
    char *bp;
    void *startp;
    vmem *v = p->next.v ;

    if ( startp = bp = vread( p, index ) ) {
        for ( i = v->ele_size; --i >= 0 ;)
            *bp++ = *this++;
        v->dirty = 1;
    }
    return startp;
}
/*-------------------------------------------------------------------------*/
PRIVATE int load_block( hdr *p, unsigned req_block ) {
/* Load a block. The base offset is in p->offset and req_block is
 * the additional offset needed to get to the correct block. */
    vmem *v = p->next.v;
    if ( req_block != v->cblock ) {
        if ( v->dirty && ! block_io ( WRITE,
                                      p->offset + v->cblock, v->buf) )
            return 0;
        if ( !block_io(READ,  p->offset + req_block, v->buf))
            return 0;
        v->dirty  = 0;
        v->cblock = req_block;
    }
    return 1;
}
/*-------------------------------------------------------------------------*/
PRIVATE int block_io( int doread, unsigned block_num, char *buf ) {
/* Read or write a block, depending on the value of doread, into or from
 * buf. block_num is the absolute block number. */
    size_t got;         /* wrote or read this many bytes         */
    long   here;        /* for debugging, make sure we got there */

    here = lseek( Vfd, (long)block_num * BLK_SIZE, SEEK_SET);
    if ( here== -1 )
        return 0;

    got = doread ? read (Vfd, buf, (size_t) BLK_SIZE)
                 : write(Vfd, buf, (size_t) BLK_SIZE) ;

    if ( got == -1 )
        return 0;

    /* it's an error if we didn't write the block, it's not an error if we
     * didn't read it, however. We may be trying to read a block at the end
     * of file that hasn't been written to yet-not a problem under UNIX but
     * it is under DOS. */

    return( doread ? 1 : (got == BLK_SIZE) );
}
/*-------------------------------------------------------------------------*/
PUBLIC long vele  ( hdr *p ) { return p->next.v->num_ele; }
PUBLIC void vdirty( hdr *p ) { p->next.v->dirty = 1;      }
#ifdef DEBUG
pfreelist() {
    hdr *p;
    if ( !(p = Freelist) )
        printf("Freelist is empty.\n");
    else {
        printf("Freelist is:\n");
        do { pheader(p);
        } while ( (p = p->next.h) != Freelist );
    }
}
pheader( hdr *p ) {
    if ( !p )
        printf("NULL");
    else
        printf("hdr   x%x: offset=%u nblocks=%u next=0x%x%s\n",
                           p, p->offset, p->nblocks, p->next.h,
                           p == Freelist ? " <-Freelist" : "" );
}
#endif /* ifdef DEBUG */
```

Listing 2

```
/* Virt.h: Prototypes are for the virtual-memory functions in virt.c. Note
 * that the hdr and char pointers have been redefined as void pointers to
 * make it easier to use the functions. */
typedef void *vhandle;

extern  int     vopen   ( void );
extern  int     vclose  ( void );
extern  vhandle vmalloc ( long num_ele, unsigned int ele_size );
extern  int     vfree   ( vhandle p );
extern  void    *vread  ( vhandle p, long index );
extern  void    *vwrite ( vhandle p, long index, void *object );
extern  long    vele    ( vhandle p );
extern  void    vdirty  ( vhandle p );
```

Listing 3

```
/* virt.hpp.  Tells the C++ compiler that these are C functions. Include
 *            this file instead of virt.h in any C++ programs. */
#ifndef __VIRT_HPP      /* take care of multiple #includes */
#define __VIRT_HPP
extern "C" {
#include "virt.h"       /* get the C prototypes */
};
#endif
```

Listing 4

```
#include <stream.hpp>
#include <stdlib.h>
#include "virt.hpp"

// VINT.CPP: Example of how to encapsulate the virtual memory functions in
//virt.c into a C++ object. #include this file to get support for virtual
//int arrays. Compile with COMPILE_METHODS defined to create library-
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
vint operator  int
int  operator  vint
vint operator  vint
```

I had originally wanted to avoid implementing the last of these, figuring that the **vint**-to-**int** conversion would take care of the right operand for me, so the first situation would do the job. This generates an ambiguity, however, because the **vint**-to-**int** conversion could just as easily be applied to the left operand as to the right. Consequently, all three situations need to be handled explicitly.

The element-selection process is in the [ ] overload function, which calls **vread** and returns a reference to itself. The operators that affect the array element modify the element selected with the previous **vread**. The **val** function just gets the current element's contents and does a little error checking. Last come the arithmetic operators. The code in Figure 2 demonstrates how the functions are used.

CONCLUSION

The subroutines and class definition in this article are not ideal, so here are some ideas for improving them. You could create a two dimensional array by creating an array of **vints**, but I wouldn't suggest it because each row would have its own buffer, so a lot of internal memory would be required. You could, of course, derive a **vint_2d** class from a **vint** to implement two-dimensional arrays directly.

Also, separating the selection process from the modification sets up a potentially risky problem. You can use a **vint** object in an expression without a previous selector, as, for example, in this code:

```
vint x(100), y(100);
x[10] = y[10];
x = y;
```

The **x=y** doesn't copy the entire array, just repeats the previous operation. The = processing code doesn't know when the element selection happened, so it uses the most recently selected elements. This is inconsistent with the way that = usually works: **x=y** should copy the y array to the x array. Because the selection process has been separated from the assignment process, **x[i] = x[j]** doesn't work. You can only select one array element at a time.

The problem could be solved by having the selector operator ([i]) evaluate to a reference to some class other than a **vint**—a derived class would be ideal. The various arithmetic operators would then work on the derived-class objects rather than on **vints**. The problem here is that a large number of these auxiliary structures would have to be created to evaluate an expression, slowing down the expression-evaluation process even more. Another solution would be to clear the **cur_ele** field every time it's used in an arithmetic operation, thus generating a run-time error when a **vint** was used without a previous index. This would slow down all array accesses, however, so I didn't implement it.

One of the main deficiencies of the current implementation is that every object has its own local buffer. This scheme can add up to a lot of buffers if there are a lot of objects. If the elements are widely separated in the array, it can also slow down transfers from one array element to another because you

CAD ENGINE

SilverScreen Licensing Program

SilverScreen is a 3D CAD/Solids Modeling software system that contains a resident C Language compiler. The C compiler was designed by Schroff Development to facilitate the development of custom applications by software developers.

SilverScreen is one of the most powerful and versatile 3D CAD/Solids Modeling systems available containing such features as: 2D and 3D Boolean operations; rendering; shading; mass properties; text editor; associative dimensioning (ANSI Standard); object-oriented database; schema-based attributes; and camera walk. SilverScreen runs on DOS and UNIX and supports a wide range of mice, video boards, printers, plotters and digitizers. File transfer is handled through DXF, IGES, HPGL, PostScript and others. Silverscreen features direct export to SmartCAM and Stereolithography (3D Systems).

SilverScreen's resident C compiler implements a library of more than 200 functions plus a large number of functions dedicated to accessing the SilverScreen database and to controlling the SilverScreen environment. Custom menus and icons also may be created. SilverScreen also adds the powerful ability to embed the SilverScreen script commands in a program for direct execution by SilverScreen.

For complete information on both the SilverScreen licensing program and other software products that have been successfully developed under this program, contact: **Schroff Development Corporation, P.O. Box 1334, Mission, Kansas 66222, 913/262-2664**

SilverScreen®

3D CAD/Solids Modeling Software

Circle 65 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
//support functions. Define MAIN to get a small test program.
//  Declarations:
//  vint x(100L);         Create a virtual array of 100 ints
//  Valid operations:
//  x[n]        This is a selector operator. The next operation
//              controls what happens to that element. The
//              following operations are supported (x is a vint
//              and i is an int):
//  i + x[n]      i - x[n]       i * x[n]       i / x[n]      i = x[n]
//  x[n] + i      x[n] - i       x[n] * i       x[n] / i      x[n] = i
//  x[n] + x[i]   x[n] - x[i]    x[n] * x[i]    x[n] / x[i]   x[n] = x[i]
//  ++x[n]        --x[n]
//  All of these operators evaluate to int rvalues, so
//  (x[i] = x[j] = x[k]) doesn't work.
//  x[n] by itself evaluates to a reference to a vint, not to an int. There
//  is a vint-to-int conversion supplied so that you can use a reference
//  directly, however.
//  Something like x+i, with no index on the x, uses the most-recently-
//  selected element. This syntax is weird--it's really a side effect of
//  the select-then-operate strategy--so you shouldn't use it.
//  Finally, note that an long-to-vint conversion is supplied by because of
//  the constructor syntax. This conversion doesn't do what you think (it
//  creates an empty array of the size specified in the conversion).
//  Be careful.
class vint {
private:
    static int nobj;            // number of active objects
    void       *hdr;            // virtual-memory handle
    int        *cur_ele;        // Most recently selected object.
    long       cur_index;       // index of "

    friend int val( vint &v );

public:
                vint          ( long array_size );
                ~vint         ();
                vint          ( vint &hdr );
                operator int  ();
          vint &operator [ ]   ( long index );
          int   operator +     ( /* this, */ int i   );
          int   operator -     ( /* this, */ int i   );
          int   operator *     ( /* this, */ int i   );
          int   operator /     ( /* this, */ int i   );
          int   operator =     ( /* this, */ int i   );
          int   operator =     ( /* this, */ vint &v );
          int   operator ++    ( /* this, */         );
          int   operator --    ( /* this, */         );
    friend int   operator +    ( int i,      vint &r );
    friend int   operator -    ( int i,      vint &r );
    friend int   operator *    ( int i,      vint &r );
    friend int   operator /    ( int i,      vint &r );
    friend int   operator +    ( vint &l,    vint &r );
    friend int   operator -    ( vint &l,    vint &r );
    friend int   operator *    ( vint &l,    vint &r );
    friend int   operator /    ( vint &l,    vint &r );
};

#ifdef COMPILE_METHODS
//constructor, starts up the virtual-memory system the first time it's
//called. Allocates a new array.
vint::vint( long array_size ) : cur_ele(NULL), cur_index(0) {
    if ( nobj++ == 0 )
        vopen();        // initialize the virtual-memory system

    if ( !(hdr = vmalloc(array_size, sizeof(int))) ) {
        perror( "VMS allocation error" );
        exit( 1 );
    }
}
vint::~vint() { // destructor, frees current object and if no more objects
    if ( !vfree( hdr ) ) {    // are left, shuts down virtual-memory system
        perror( "VMS deallocation error" );
        exit( 1 );
    }
    if ( --nobj == 0 )
        vclose();           // close down the virtual-memory system
}
vint::vint( vint &src ) : cur_ele(NULL), cur_index(0) {
// Copy constructor. VERY expensive operation. duplicates array in src.
    int *p;
    long numele = vele( src.hdr );

    if ( nobj++ == 0 )
        vopen();            // initialize the virtual-memory system

    if ( hdr && !vfree( hdr ) ) {
        perror( "VMS copy-deallocation error" );
        exit( 1 );
    }

    if ( !(hdr = vmalloc( numele, sizeof(int))) ) {
        perror( "VMS copy-allocation error" );
        exit( 1 );
    }

    while ( --numele >= 0 ) {
        if ( !(p = vread( src.hdr, numele )) ) {
            perror ( "VMS copy-access (read) error." );
            exit( 1 );
        }

        if ( !vwrite( hdr, numele, p ) ) {
            perror( "VMS copy-access (write) error.\n" );
            exit( 1 );
        }
    }
}
vint::operator int() {      // convert vint to int
    if ( cur_ele )
        return *cur_ele;

    cerr << "VMS: no previous index on vint.\n";
    return 0;
}
vint &vint::operator [ ]( long index ) {  // select an array member
    if ( !(cur_ele = vread( hdr, index )) )
        perror( "VMS selection error" );

    cur_index = index;
    return *this;
}
// The following methods all modify the member that was selected by a
// previous array-index ([i]) operator. Note that there's no provision for
// a vint=vint operator because the earlier vint-to-int conversion will act
// on the right argument to make it suitable for the vint=int operator
// defined below.
static int val( vint &v ) { // return the value of the currently-
    if ( v.cur_ele )        // selected member.
        return *v.cur_ele ;

    cerr << "VMS: No previous index on vint (val)\n" ;
    return 0;
}
int vint::operator ++ () {  // increment the current member
    val(*this);             // val call for error checking.
    vdirty(hdr);
    return ++*cur_ele;
}
int vint::operator -- () {  // decrement the current member
    val(*this);             // val call for error checking.
    vdirty(hdr);
    return --*cur_ele;
}
int vint::operator = ( int i ) { // assign to the current member
    if ( !cur_ele ) {
        cerr << "VMS: No array index on a vint (=)\n" ;
        return 0;
    }
    vdirty( hdr );
    return *cur_ele = i;
}
#endif /* COMPILE_METHODS */
// These functions take care of the nondestructive arithmetic
// operations. The ones with two arguments are friends of class vint.
inline int vint::operator +    ( int i ){ return  val(*this) + i; }
inline int vint::operator -    ( int i ){ return  val(*this) - i; }
inline int vint::operator *    ( int i ){ return  val(*this) * i; }
inline int vint::operator /    ( int i ){ return  val(*this) / i; }

inline int operator + (int i,   vint &r){ return  i      + val(r); }
inline int operator - (int i,   vint &r){ return  i      - val(r); }
inline int operator * (int i,   vint &r){ return  i      * val(r); }
inline int operator / (int i,   vint &r){ return  i      / val(r); }

inline int operator + (vint &l, vint &r){ return  val(l) + val(r); }
inline int operator - (vint &l, vint &r){ return  val(l) - val(r); }
inline int operator * (vint &l, vint &r){ return  val(l) * val(r); }
inline int operator / (vint &l, vint &r){ return  val(l) / val(r); }
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 2: Using the vint type

```
main ()
{
    const int size = 5;
    int result;
    int i;
    vint x = (long)size;      //array of "size" ints
    // load each element of the array with its own index
    // then read the array back to verify
    for (i=size; --i >= 0; x[i] = i) ;
    for (i=size; --i >= 0; )
        if ( (result = x[i]) != i )
            cout << "x[" << i << "]==" << result << "\n";
// legal operators---------------->|          |<----------
    cout << "x[3] = 3     = " << (x[3]   = 3     ) << "\n" ;
    cout << "x[3]         = " << (x[3]           ) << "\n" ;
    cout << "x[3] + 5     = " << (x[3]   + 5     ) << "\n" ;
    cout << "x[3] - 5     = " << (x[3]   - 5     ) << "\n" ;
    cout << "x[3] / 5     = " << (x[3]   / 5     ) << "\n" ;
    cout << "x[3] * 5     = " << (x[3]   * 5     ) << "\n" ;
    cout << "5    + x[3]  = " << (5      + x[3]  ) << "\n" ;
    cout << "5    - x[3]  = " << (5      - x[3]  ) << "\n" ;
    cout << "5    * x[3]  = " << (5      * x[3]  ) << "\n" ;
    cout << "5    / x[3]  = " << (5      / x[3]  ) << "\n" ;
    cout << "x[3] + x[3]  = " << (x[3]   + x[3]  ) << "\n" ;
    cout << "x[3] - x[3]  = " << (x[3]   - x[3]  ) << "\n" ;
    cout << "x[3] * x[3]  = " << (x[3]   * x[3]  ) << "\n" ;
    cout << "x[3] / x[3]  = " << (x[3]   / x[3]  ) << "\n" ;
    cout << "++x[3]       = " << (++x[3]         ) << "\n" ;
    cout << "--x[3]       = " << (--x[3]         ) << "\n" ;
}
```

would have to do a lot of swapping. A solution would be a somewhat more sophisticated buffering scheme. A pool of buffers could be assigned to objects on an as-needed basis, at the first read or write request. Rather than swapping out a bufferful of data to read in an array element, an additional buffer would be assigned to the object at that point.

The buffers could be arranged in a linked list, the head of which is in the **vmem** structure. The *buf* field of the **vmem** structure is a pointer rather than being the actual buffer to allow for this change. Each buffer would be time stamped every time it was accessed, and when no more buffers were available in the pool, the buffer with the earliest time stamp could be swapped out and assigned to the new element. Objects that were accessed more often would tend to have several buffers, while rarely-accessed objects could have no buffers assigned to them at all until necessary.

A final issue (which wouldn't be an issue if you made the buffer changes) is the current block size. I've made it 256 bytes as a compromise. Since each object has its own buffer, the buffer had to be reasonably small or the objects would require too much space. On the other hand, larger buffers tend to make for more efficient reads and writes. On UNIX, a block size of 512 makes a lot more sense than 256 because that's the disk-block size. Even on MS-DOS, a bigger buffer would give you slightly faster read times. A bigger buffer would also give you a higher "hit rate," since the odds of a desired array element's being in memory would be higher. If you are only using a few virtual objects, you should probably redefine BLK_SIZE to increase the swap-buffer size.

The foregoing notwithstanding, the current implementation is pretty useful. It does a complex task relatively painlessly, and frees the programmer from having to worry about the mechanics of virtual memory in order to use the disk to store active data. The routines are easy to modify for other types, and the use of additional operators is not complicated.

. . lean, mean, programming machine . .

Q-PRO 4 is the ***killer*** 4GL for applications development. 44k of lean, mean, compact, professional quality, industrial grade, field proven (over 25,000 installed) interpreter. It runs on DOS, LAN's, ConDOS, PCMOS-386 (with *record and file lock*). Q-PRO 4's execution speed and capabilities rival compiled C but only 1/100th the programming time! Proof:

Q-PRO 4's Lightning-Fast File and Screen Handling!

5 min. 56 sec.	Write 10,000 unique 255 byte records and update 4 indices.
24 sec.	Add 100 new unique records at random locations.
0.2 sec.	Position to a record in the file by its primary index; display the record's 50 fields formatted on the screen.

Q-PRO 4's Efficient Syntax for Fast Development!

3 Commands	Retrieve a record using its key and display all its fields to the screen.
2 Commands	Write a full screen (up to 1900 fields) to an indexed file and update all keys.
0 Commands	Build a complete data entry screen with fill-in-the-blanks entry fields, prompts and field edits (justification, type checking, must tab, etc.).

Q-PRO 4's Complete Features !

Fabulous File Handling	255 files open at one time. 9 indices per file. Does 1-to-1, 1-to-many and many-to-many relationships easily. Data dictionary.
Unique Screen Control	Table driven screen with full color control. Multiple data entry screens per program module. Dialog boxes. Lightbar menus. More.
Complete Command Set	Over 110 commands and functions. Unique keyed text notepad system and context sensitive help. And, it's extensible -- add your own routines in C or assembler!
Report Generator	Up to 6 files. Re-relate at report time. Math, totals, subtotals, control breaks.
LockUp System	Encrypts and locks your code for distribution. No royalties! (Optional)
Query Language	Post relational. Up to 12 files in database. Select, sort, total, high, low, etc. Mailing label generator. (Optional)

Order Q-PRO 4 today! Only $200.00 (+7.75 S&H) until 12/31/90. 30 money back guarantee. Check, VISA, MC, COD. Demo and evaluation systems available.

QNE International®

136 Granite Hill Court • Langhorne, PA 19047
TEL: (800) 333-0448 FAX: (215) 968-3550
(All timings om 8Mhz AT clone, 30 ms disk.)

Circle **60** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Tom Swan

Pitfalls in Turbo Pascal OOP

Some potential traps discovered in recent Pascal OOP expeditions

To gain a better footing with Turbo Pascal objects, I've documented six pitfalls that trapped me in recent OOP expeditions. If you're struggling to conquer object-oriented concepts, this article may help you avoid falling into the same holes where I've tumbled.

The following experiences come from VA ("View Anything"), an object-oriented file viewer that I've been sharing piecemeal in *PJ*. Although the code is nearly complete, the source files have grown larger than are practical to publish here. Watch for a future notice about how to get a copy of the finished program, or send your name and address to Swan Software, P.O. Box 206, Lititz, PA 17543. I'll notify you when VA is done.

THE FIRST TRAP

Failing to initialize the VMT is the first of my six pitfalls. If you've read the Turbo Pascal 5.5 OOP Guide, you probably know that before calling an object's virtual methods, you must call its constructor. That links the object instance (a variable of an object type) to the appropriate VMT (Virtual Method Table), which stores virtual method addresses. If you don't perform this step, virtual method calls will be directed to random memory locations. As you might expect, this usually causes the program to crash.

Many people are confused about which object types have associated VMTs, which objects need constructors, and what the relationship is between constructors and virtual methods. I find it helps to keep these facts in mind:

- The compiler creates a new VMT for any object type that declares or inherits at least one virtual method or constructor.
- Without exception, an object that declares or inherits one or more virtual methods must have or inherit a constructor.
- An object may have a constructor and no virtual methods. Even so, you must call the constructor to initialize an object before performing other actions that consult the object's VMT—for example, assigning one object instance to another.

Also remember that there is only one VMT per object type. Because all object instances of the same type share a single VMT, beginners often assume they can call an object's constructor once and then forget about it. Uh uh. You must call the constructor to link every object instance to the correct VMT even if all of those instances are of the same type.

COPYING OBJECTS

Using Pascal's assignment operator to copy one object to another can lead to serious trouble. To understand the problem, consider what happens when copying a static object to another static object of the same type. (A static object is one that you define with **Var**.) To simplify the code, I'll represent various data fields here as type **ObjectFields**. Suppose you declare:

```
Vehicle = object
  data : ObjectFields;
  constructor init( params : ObjectFields );
end;
```

After defining two variables *v1* and *v2* of type **Vehicle**, if you want to copy *v1* to *v2*, you may be tempted to write:

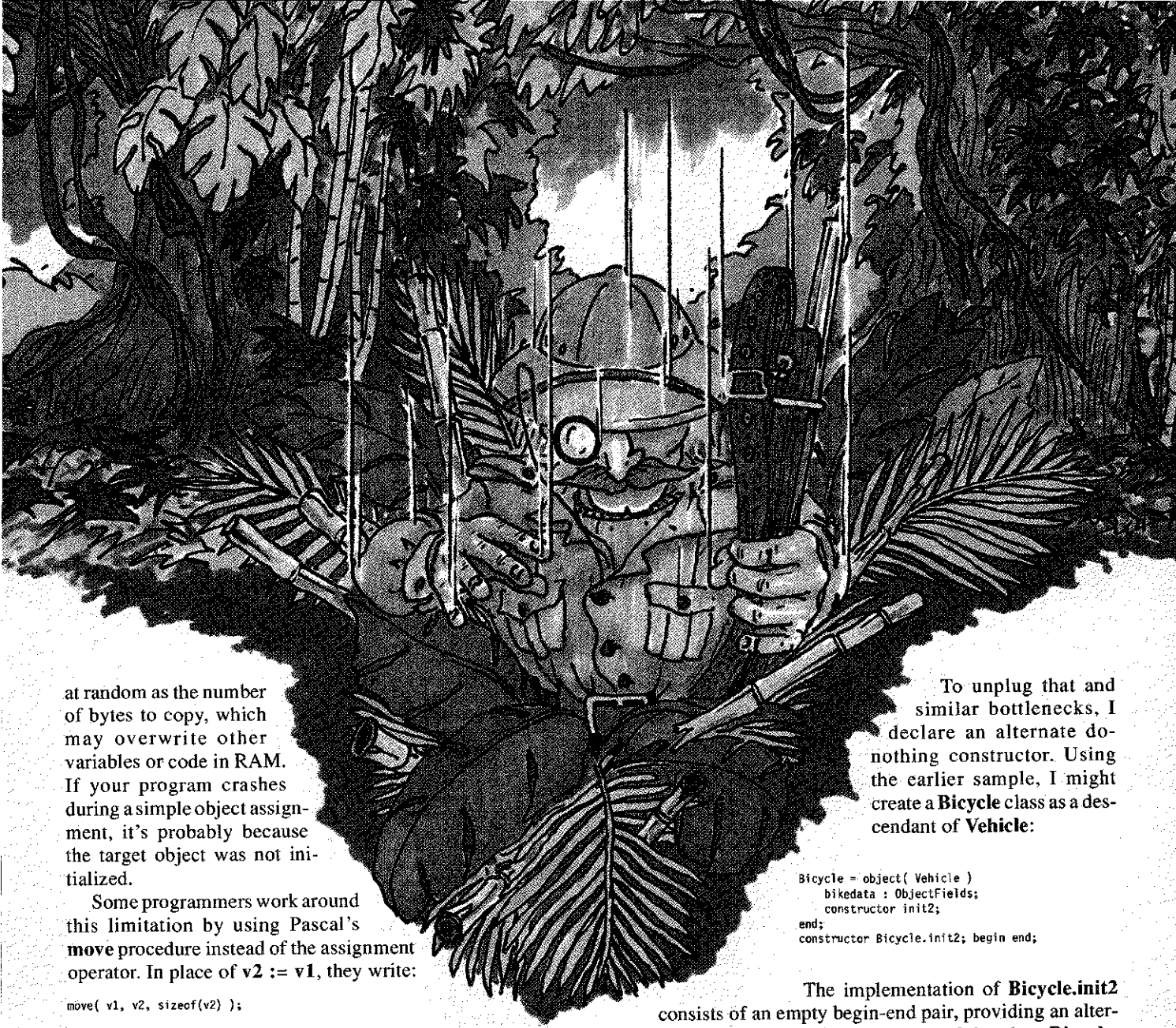
```
v1.init( args );      { Initialize object v1 }
v2 := v1;             { ??? Assign v1 to v2 }
```

Despite appearances, that innocent-looking assignment may cause the program to crash. Because Turbo Pascal consults the target object's VMT to determine how many bytes to copy from the source, and because the target (*v2* in this example) is not yet linked to its VMT—which requires calling *v2*'s constructor—the run-time copy routine that carries out assignments can't determine *v2*'s correct size. Worse, instead of halting with an error, TP's run-time code simply uses a value

---

Tom Swan is a contributing editor of Programmer's Journal *and the author of* Mastering Turbo Debugger *(1990),* Mastering Turbo Pascal 5.5 *(1989), and* Mastering Turbo Assembler *(1989), all published by Howard W. Sams, Indianapolis, Indiana. Send mail to Tom in care of* PJ, *or to his CompuServe ID: 73627,3241.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



at random as the number of bytes to copy, which may overwrite other variables or code in RAM. If your program crashes during a simple object assignment, it's probably because the target object was not initialized.

Some programmers work around this limitation by using Pascal's **move** procedure instead of the assignment operator. In place of **v2 := v1**, they write:

```
move( v1, v2, sizeof(v2) );
```

This copies all bytes from *v1* to *v2*, including *v1*'s 2-byte link to its VMT; therefore, the target object *v2* does not have to be initialized. However, because this causes objects *v1* and *v2* to address the same VMT, this technique works only with objects of identical types. Never use **move** to copy objects of different types.

Instead of resorting to **move**, the best (but not the most efficient) solution is to initialize the target object before the assignment:

```
v1.init( args ); { Initialize object v1 }
v2.init( args ); { Initialize object v2 }
v2 := v1;        { v1 is a descendant of or same class as v2 }
```

This also works for objects of different types—that is, as long as the target object is an ancestor of the source. Initializing object *v2* before copying lets the run-time copy routine find the correct size of the target from that object's VMT. It then copies that many bytes from the (usually) larger *v1* to the (usually) smaller *v2*. Unfortunately, passing arguments to *v2*'s **init** constructor is wasteful when all you want to do is copy initialized fields from *v1* to *v2*. Because you have to call the constructor before the assignment, the same fields in *v2* are initialized twice—once by the constructor, and again by the assignment.

To unplug that and similar bottlenecks, I declare an alternate do-nothing constructor. Using the earlier sample, I might create a **Bicycle** class as a descendant of **Vehicle**:

```
Bicycle = object( Vehicle )
    bikedata : ObjectFields;
    constructor init2;
end;
constructor Bicycle.init2; begin end;
```

The implementation of **Bicycle.init2** consists of an empty begin-end pair, providing an alternate way to initialize object instances of the class **Bicycle**. The alternate constructor does not prepare **Bicycle**'s data fields, but it does establish the link between a **Bicycle** instance and its VMT. Using this setup, if **bike1** and **bike2** are **Bicycle** objects, I can initialize one object and assign it to the other with these statements:

```
bike1.init( args );   { Initialize Bicycle #1 object }
bike2.init2;          { Partially initialize Bicycle #2 object }
bike2 := bike1;       { Assign same-type object bike1 to bike2 }
```

This approach is more efficient than the previous one because it passes arguments to the constructor only once. The design is also useful for copying dissimilar objects. Assume **bike** is a **Bicycle** (the descendant), and **scooter** is a **Vehicle** (the ancestor). Then after adding a do-nothing **init2** constructor to the **Vehicle** class, it's possible to copy dissimilar but related objects this way:

```
bike.init( args );    { Initialize the bike object }
scooter.init2;        { Partially initialize scooter }
scooter := bike;      { Assign descendant bike to scooter }
```

The assignment of **bike** to **scooter** copies only the data fields in the **bike** descendant that are shared by the ancestor object—in other words, only the stuff that is **scooter**-like in a **bike**. It

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

does not copy the VMT link from **bike** to **scooter**. Though related, the two objects have separate VMTs. The ancestor's do-nothing **init2** constructor establishes the link between **scooter** and **Vehicle**'s VMT. Because **init2** does nothing else, it runs quickly. This improves performance and allows the assignment of **bike** to **scooter** (descendant to ancestor) because the run-time copy routine can now determine **scooter**'s size from **Vehicle**'s virtual table.

Related note: When objects contain pointer fields, copying one object to another may cause two pointers to address the same data in memory. That's okay as long as you don't dispose that same memory more than once. Guard against this error by setting pointers to **nil** after disposing their allocated space and checking for **nil** pointers before calling **dispose** (usually in the object's destructor).

CALLING VIRTUAL METHODS

A constructor may call static methods in its own or in ancestor classes. A constructor may also call virtual methods that belong to its own, an ancestor's, or even a descendant's classes. To understand how a constructor can call a method in a descendant class that hasn't yet been declared, remember from earlier that you must call an object's constructor before using any of that object's virtual methods. Because the constructor's entry code links the object and its VMT, it's always safe to call virtual methods from inside the constructor itself.

Or, rather, it's almost always safe. I wasted hours hunting a bug caused by a virtual method call inside a constructor. I caused the problem when I failed to consider whether to initialize local data fields before or after calling the ancestor object's constructor. To demonstrate how this boxed me into a canyon, suppose **Vehicle** is declared as:

```
Vehicle = object
  vehicledata : ObjectFields;
  constructor init;
  procedure prepareData; virtual;
end;
```

The implementation of the **init** constructor calls the virtual method **prepareData**, initializing data fields inside the object, and then performing other actions affecting local data in a **Vehicle** object:

```
constructor Vehicle.init;
begin
   prepareData;
   { ... other actions }
end;
```

Now, consider what happens if a descendant inherits fields and methods from **Vehicle** and replaces the virtual **prepareData**. Since this is a racier idea, let's name the new class **MotorBike**:

```
MotorBike = object( Vehicle )
  motorData : ObjectFields;
  procedure prepareData; virtual;
end;
```

MotorBike declares additional fields, represented here by **motorData**. It also declares a replacement **prepareData** method, which, as it does in the ancestor, prepares fields in the object. Notice that the descendant does not declare its own constructor—it inherits the one from **Vehicle**. This is where the tricky part comes in. Using pseudocode for illustration, here's a possible design for **prepareData**:

```
procedure MotorBike.prepareData;
begin
   motorData := initialValues;  {Other duties}
   Vehicle.prepareData;
end;
```

The decision about which of the two **prepareData** methods to call is made at run time (in a process called "late binding"), and depends on whether the object is a **Vehicle** or a **MotorBike**. If the object is a **Vehicle**, the original **prepareData** method runs. If it's a **MotorBike**, the replacement runs, which then calls the ancestor's method. This also holds true for calls to **prepareData** from inside the constructor, allowing **dirt-**

Complete Modem Communications

Using Sugar Coated Serial™

Now for both **C** and **Turbo Pascal** Programmers

Features:

- Complete pull-down menu communications program and its commented source code using only SCS routines.
- Supports XModem, YModem, ASCII, & Telink protocols.
- All source code included including assembler, with many predefined constants that take the pain out of understanding and modifying nitty-gritty, low-level aspects of communications.
- No knowlege of assembly language is required...procedure calls are as simple as "MakeCall(PhNumStr)" and "SendFilesYModem.
- Versions for TP v5.0 or greater and Microsoft & Turbo C.
- Complete user manual including an informative and practical discussion on the theory of serial communications.
- No royalties on programs you sell
- Interrupt driven with no TSR requirement.
- Pull-down menu source library included.

Complete system	$109.95	(For Turbo Pascal or C)
Demo Disk	$5.95	($6.95 for 3½" disks)
Information pak	Free	

Shipping included, IL residents add 6.25% sales tax. Please specify disk size and language (C or Turbo Pascal.)

For credit card orders or info call **(708) 759-1270** or send check or money order to:

Goin Places Software
Suite 145
1202 N. West 75th Street
Downers Grove, IL 60516

Visa MC

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"... one of those rare events where the talks were actually useful for *real* problems and *real* jobs."

—William Roetzheim, *Programmer's Journal*

This Fall, For the First Time ... in Boston.

This Fall, a West Coast tradition will encore in the East. Software Development, the three-day conference and trade show, comes to the Omni Parker House in Boston, Nov. 6-9.

SOFTWARE DEVELOPMENT '90—FALL is a different kind of conference, created by and for programmers. You'll have three days to focus on practical techniques that will increase your programming power. You'll learn from the most important visionaries and practitioners in software development today. Speakers include: Larry Constantine, P.J. Plauger, Jack Purdum, John Bentley, Bruce Eckel, Steve Weiss, Robert Ward, Tom Plum, David Schmitt, Jim McCarthy, Meilir Page-Jones, and more.

SOFTWARE DEVELOPMENT '90 — FALL features the most popular classes from the West Coast conference, plus a number of new classes taught by distinguished speakers from the East Coast.

Topics include object-oriented programming, programming in C, design methodologies, and management issues. All sessions are tailor-made for you — the experienced software developer.

Share tips and techniques with the "best and the brightest" at informal evening receptions and vendor hospitality events.

And the very latest in development tools and services will be on display at the concurrent Software Development Products Exhibition.

SOFTWARE DEVELOPMENT

Mark Your Calendar:

Software Development '90—Fall
November 6-9, 1990
Omni Parker House,
Boston, MA

Software Development '91
February 12-15, 1991
Santa Clara Convention Center,
Santa Clara, CA

Class sizes are limited and discounts are available for early registration. Call (415) 995-2472 for more information.

Software Development '90—Fall is sponsored by Miller Freeman Publications, publishers of *Computer Language, AI Expert, Embedded Systems Programming, UNIX Review, Database Programming & Design,* and *LAN Magazine.*

Software Development Seminars,
500 Howard St., San Francisco, CA 94105
Call: (415) 995-2472 Fax:(415) 267-7683

"This is the third time I have gone (to the SOFTWARE DEVELOPMENT Conference). In one word: it is the finest programming conference offered, bar none."

Andrew Binstock, editor,
The C Gazette, Santa Monica, CA

"...SOFTWARE DEVELOPMENT covers all the bases. ...this will clearly be a "hot" conference over the next few years."

Tom Schwartz, contributing author,
IEEE Expert, Los Alamitos, CA

"Far exceeded my expectations. It's great sharing time and information with real experienced, creative people — real answers without the hype. Probably the most valuable thing I've done in 10 years in the software business."

Chuck Atkinson,
CAP, Fort Worth, TX

"Great Conference. In just four weeks the conference has already paid for itself. I'll definitely be back!"

Risa Hyman,
Arinc Research, Annapolis, MD

❑ Send me more information on Software Development '90 — Fall, November 6-9, 1990, at the Omni Parker House, Boston, MA

Name _____

Title _____

Company _____

Address _____

City, State, Zip _____

Phone/FAX _____

Send me information on: ❑ attending ❑ exhibiting PJ9

Software Development Seminars, 500 Howard St., San Francisco, CA 94105
Call: (415)995-2472 Fax:(415) 267-7683
Circle 167 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Hard-core OOPers say that "Abstract objects are never instantiated."

Bike.init; to initialize a **dirtBike** (which will be an instance of type **MotorBike**) even though that class does not provide its own constructor.

The call to **init** causes the inherited constructor to call **MotorBike**'s replacement **prepareData** for **dirtBike**. While useful, this can lead to bugs when the replacement **prepareData** method inside **MotorBike** depends upon ancestor fields being initialized in the object. (See the location marked by "{Other duties}" earlier.) In that case, the method must call **Vehicle.prepareData** before performing other actions, not afterward. In general, that's a good rule to follow: initialize ancestor fields by calling the ancestor's constructor before performing local initializations inside the descendant. That way, you'll be sure that descendant objects play with all their (initialized) marbles.

REPLACING ABSTRACT METHODS

An abstract object is similar to a scale model of a building. The model shows the layout of rooms and hallways, but it doesn't have elevators or a sprinkler system. Since an abstract object is only a model, you will never declare variables of the abstract data type. (Hard-core OOPers say that "abstract objects are never instantiated.") Instead, you'll declare a descendant object that inherits from an abstract ancestor class, and then define variables of the descendant data type.

Another characteristic of an abstract object is that it usually contains virtual do-nothing methods. Not surprisingly, these placeholder procedures and functions are known as "abstract methods." Problems frequently arise when descendant objects neglect to provide replacement procedures or functions for abstract virtual methods. For example, suppose you declare an object class named **AbstractObject**:

```
AbstractObject = object
  data : ObjectFields;
  constructor init( d : Objectfields );
  procedure doSomething; virtual;
end;
procedure AbstractObject.doSomething;
begin
  (* TO BE SUPPLIED BY A DESCENDANT OBJECT *)
end;
```


HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

MEET THAT DEADLINE ! ! !

- Combine the raw power of extensible languages with the convenience of carefully implemented functions!
- Execute faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Modify routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement that borrowed feature, often more efficiently than in its source!
- Use an architecture that supports very small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Use subroutine libraries written for other languages! More efficiently!
- Outproduce experienced programmers stuck using conventional languages!

HS/FORTH with FOOPS - The only full multiple inheritance object oriented language under MSDOS!

WAKE UP ! ! !

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly.

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful only in learning Forth's basics, they fail at showing its true potential. Not to mention being s-l-o-w.

You can't add extensibility to fossilized compilers. You can easily add features from any other language to **HS/FORTH**. Try our full modifiable multiple inheritance **OOPs** extension or our **Fortran** translator & mathpak!

HS/FORTH has more features and libraries than we can possibly describe here. **Please ask for our brochure!**

HS/FORTH uses MSDOS or PCDOS. Each level includes features of lower ones. Level upgrades: $25. plus price difference. Sources in ordinary text files.

STUDENT LEVEL $145.
text & scaled/clipped graphic windows, ellipses, splines, arcs, fills, turtles; powerful parsing, formatting, file and device I/O; shells; interrupt handlers

PERSONAL LEVEL $245.
software floating point, trig, transcendental, quad & scaled integer, automatic optimizer for assembly like performance.

PROFESSIONAL LEVEL $395.
hardware floating point, turnkey, seal, Rosetta Stone Dynamic Linker, round robin & time slice multitaskers, dynamic string manager, file blocks, sector mapped blocks, assemblers

PRODUCTION LEVEL $495.
Metacompiler: DOS/ROM/direct/indirect C data structures & struct compiler, TurboWindow-C graphics library,

Options:

FOOPS+ with multiple inheritance **$ 75.**
286FORTH or **386FORTH** **$295.**
TurboWindow-C (Metagraphics) **$ 99.**
BTRIEVE for HS/FORTH (Novell) **$199.**
ROMULUS HS/FORTH from ROM **$ 95.**
FFORTRAN translator/mathpak **$ 75.**

Circle 42 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The **doSomething** virtual method is abstract—its only purpose is to inform the compiler that all descendants of **AbstractObject** will have a **doSomething** method. A descendant object is expected to inherit properties from **AbstractObject** and provide a replacement **doSomething** method. In its new incarnation, **doSomething** will then do whatever needs doing.

Serious bugs may arise if a descendant object neglects to provide a replacement for an abstract method like **doSomething**. For example, a graphics object may declare a **drawLine** abstract method. Unless a descendant class provides a working **drawLine** replacement, the graphics library will be unable to draw lines.

To avoid our fourth pitfall and prevent such errors, I call a simple error procedure from inside most abstract methods. I name my procedure **abstractErr**, but what you call it isn't critical. (Some of Borland's sample OOP programs provided with Turbo Pascal 5.5 include a similar procedure, which they called **abstract**.) The formerly empty **doSomething** now becomes:

```
procedure AbstractObject.doSomething;
begin
  abstractErr( 'AbstractObject.doSomething' )
end;
```

Somewhere else, perhaps in a unit of common routines, I implement the **abstractErr** procedure:

```
procedure abstractErr( s : string );
begin
  gotoxy( 1, 25 );
  writeln;
  writeln( 'Error: Abstract method called' );
  writeln( '       in ', s );
  halt( 1 )
end;
```

If any object fails to provide proper replacements for abstract methods, the error handler will halt the program with a message describing the location of the problem. This may seem to be an unfriendly act of war on program users, but it's not something that would happen in completely debugged, finished code. (Right?)

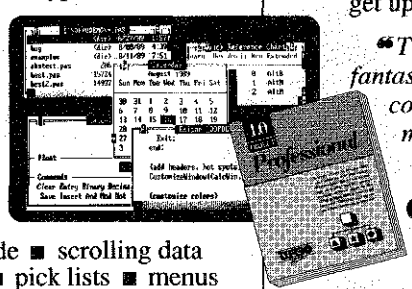
In practice, this technique is useful primarily during early design stages when you are inheriting numerous objects, providing replacement methods, and declaring new abstracts. Instead of forcing you to suffer through mysterious bugs caused by calls to abstract methods that you forgot to supply, **abstractErr** immediately halts the program so you can complete critical methods you missed.

You don't need to take such drastic measures for all abstract methods. To use another graphics example, an object might declare an abstract **fillPattern** method, which you are free to provide or not as you choose. If you declare a replacement

Save "Man-Years of Effort" with Turbo 5.5

Don't Start from Scratch with Object-Oriented Pascal

Object Professional is a huge library of over 200 object types and 2000 methods that will multiply your productivity. Window object types let you use overlapping and resizeable windows. The windows include ■ scrolling data entry screens ■ pick lists ■ menus ■ file selection ■ printed forms ■ help capability and more.



Build your programs using proven data object types like stacks, linked lists, virtual arrays, and more. System-oriented routines provide swappable TSRs in only 6K of RAM, EMS management, and much more.

Object Professional includes clear, comprehensive documentation, on-line help, full source code, technical support, and hot demo programs. Pay NO royalties. You'll get up to speed fast with OOP!

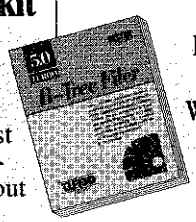
"The range of objects is fantastic. Object Professional could literally save you man-years of effort."

Jeff Duntemann

Object Professional 1.0, only $150.

A Multi-User B-Tree Toolkit

Write powerful network compatible databases faster and easier using **B-Tree Filer 5.0**. You'll have the fastest, safest, most flexible databases – no rigid structure, no TSR hassles, no running out of files. And they're compatible with Novell, 3Com, MS-NET, and others.



You get ■ Fixed and variable length records ■ Two billion records per database ■ Up to 100 indexes per index file ■ Fail-safe mode with journaling ■ Units for sorting, browsing, reindexing, and network control.

B-Tree Filer includes full source code, documentation, technical support, and you pay NO royalties.

"B-Tree Filer... a well rounded, feature-rich approach to B-Tree databases."

Computer Language, 1/90

B-Tree Filer 5.0, only $125. (single user)
With network support, $175.

Call toll-free to order.
1-800-333-4160

Satisfaction guaranteed or your money back within 30 days.
Add $5 per order for shipping in U.S. and Canada. Inquire about other shipping charges. OPro requires Turbo 5.5. BTF requires Turbo 4.0, 5.0, 5.5, or QuickPascal.

TURBO Power

8AM - 5PM PST Monday through Friday, USA & Canada.
For more information call (408) 438-8608. Fax: (408) 438-8610.
TurboPower Software PO Box 66747 Scotts Valley, CA 95067-0747

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

fillPattern, then shapes will be filled. If you don't declare a replacement, the program's fill routines will call the abstract method, but since it does nothing, shapes remain unfilled. This is a harmless action that probably doesn't deserve to be treated as a fatal condition.

THE FIFTH TRAP

OOP purists usually frown (scowl, really) at programming that reads or writes data fields inside objects. Others say that's hogwash—programmers should be free to access object data directly when doing that adds speed to code.

As with most controversies, both sides have good points. If there's any chance that a descendant object will change the data types or storage methods of ancestor fields, then you should use virtual methods to read and write those values. If not, then go ahead and break the rule. In other words, you are supposed to declare objects this way:

```
intArray = array[ 1 .. maxInt ] of integer;
Vector = object
  maxIndex : word;   { Highest allowed index }
  aPtr : ^intArray;  { Pointer to an array of integers }
  constructor init( n : word ); { Create array[1..n] of integer }
  function get( index : word ) : integer; virtual;
  procedure put( value : integer; index : word ); virtual;
end;
```

In this sample, a **Vector** object contains a pointer to an array of integers. Although not shown here, and ignoring error handling, the **init** constructor allocates enough memory to hold *n* integers, assigning the address of that memory to *aPtr*. Methods **get** and **put** assign and retrieve values from a **Vector**. The *aPtr* and *maxIndex* fields in this example are in hands-off territory.

Because all accesses to **Vector** objects are through the **get** and **put** virtual methods, descendant objects may replace those methods to implement an array of integers using a different storage technique. For example, a descendant could store arrays in EMS memory or on disk. Other calls to **get** and **put** will be redirected at run time to the replacement methods, thus altering the way data is stored, but not changing the way that data is used.

Turbo Pascal does not provide a way to restrict access to fields like *aPtr* and *maxIndex* for the private use of an object's own methods, though, so programmers are free to read and write an object's fields directly without being chewed out by the compiler. But then, if a descendant object changes the way **Vector**s are stored, *aPtr* will no longer address valid data and the modified code will fail.

However, if a descendant will never provide a different data storage mechanism, then the direct access to private data in objects may boost the program's performance. So, it's a tradeoff. You just have to decide whether to follow the purists and use methods to access data fields, or break the rules and access object data directly to get better performance. At least Turbo Pascal lets you make that choice, even if it does nothing to prevent you from choosing badly and, in the long run, making more work for yourself down the road.

THE WRONG TYPE CAST

Movies may survive bad casting, but not object-oriented programs. A typical type-cast problem occurs in a statement that retrieves a pointer to an object that you know is really a descendant of that object's type. As explained earlier, it's perfectly okay for a pointer to an ancestor to address a descendant object because the descendant shares all the properties of its ancestors. Whatever data fields are available through the ancestor pointer are also available in the descendant. But to gain access to the descendant's additional fields, you must use a type cast that tells the compiler to assume that the pointer addresses an object of the descendant class.

Typically, this situation comes up when an object declares a function method that returns a pointer to an ancestor type. Often, a descendant will use that method to refer to other descendant objects, requiring a type cast to transform the function result into a pointer to the descendant class. A sample demonstrates the problem:

```
List = object
  data : ObjectFields;
  constructor init;
  procedure insertItem( i : ItemPtr );
  function currentItem : ItemPtr;
  { other methods }
end;
```

If you've been following these articles, you'll recognize that bit of code as a loose approximation of VA's **List** class. In addition to its data fields, the constructor **init** prepares a new **List** instance. Method **insertItem** inserts new items into a **List**. Method **currentItem** returns a pointer to the currently addressed item. Other methods move to the next and previous items, delete items, dispose **List** objects, and so on.

Take note of the fact that **currentItem** returns a pointer of type **ItemPtr**, which addresses an abstract **Item** object (not shown here). Typically, a program will declare a descendant of **Item**, then call **insertItem** to add new non-abstract objects to a list.

Then, however, to access those items with **currentItem** requires a type cast that tells the compiler what **currentItem** actually addresses. For example, if **NewItem** descends from **Item**, this calls method **NewItem.NewMethod** in a listed object:

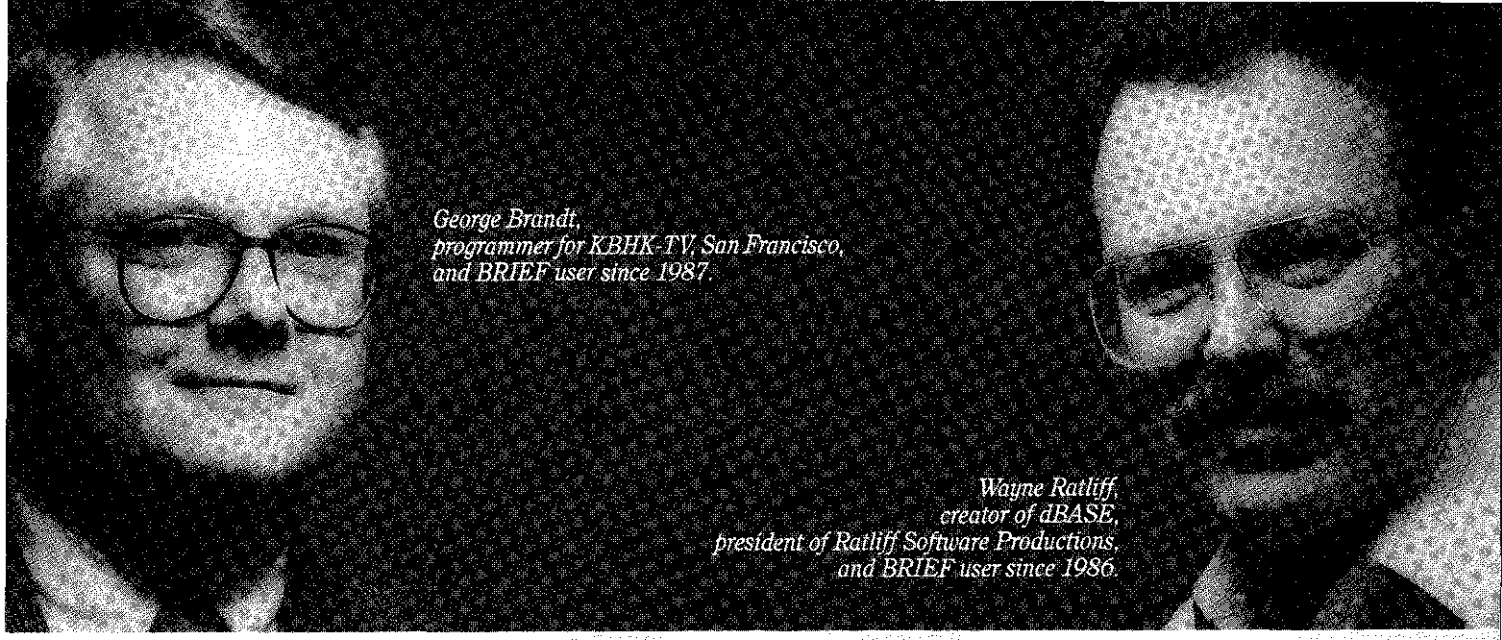
```
NewItemPtr( theList.currentItem )^.NewMethod;
```

Placing **theList.currentItem** inside parentheses preceded by **NewItemPtr** (declared as **^NewItem**) casts **currentItem**'s data type **ItemPtr** to **NewItemPtr**. This tells the compiler that **currentItem** actually addresses a **NewItem**, not an **Item** object. Knowing that, the compiler then allows the pointer dereference (^) and it generates the necessary code to call **NewMethod**.

The need for type casting is one of OOP's most bug-prone features. If you use the wrong cast expression, the compiler may generate calls to non-existent methods, access fields out of thin air, and attempt to perform other magic tricks that can blow up in your face. Because the compiler provides no protection against bad casting, you should use type cast expressions with extreme care.

I suppose rules like that are to be expected of sophisticated concepts like OOP, which requires programmers to step carefully in order to avoid pitfalls. Documenting six of the most common OOP faults that have given me trouble in the past may not prevent future cracks in my code, but at least it's helping me to stay out of the same old holes. PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



George Brandt,
programmer for KBHK-TV, San Francisco,
and BRIEF user since 1987.

Wayne Ratliff,
creator of dBASE,
president of Ratliff Software Productions,
and BRIEF user since 1986.

Of course they need completely different kinds of editors. That's why they both use BRIEF.

Let's face it. There's no such thing as two programmers who think alike. Or who work alike.

Because everybody's got their own style, their own habits. Their own quirky little requirements. As well they should. So, where are you going to find one editor that satisfies everyone? Simple. Look inside a BRIEF® box.

It's the right editor, right out of the box.

What you'll find inside is an enormously powerful, astonishingly easy, and highly-evolved program editor. One that works at blinding speed. And that's loaded with features designed to help you be more productive. One guaranteed to give you excellent results within one short hour of booting up the first time.

You'll find features like automatic indenting, infinite windowing, and powerful regular expression searching that goes well beyond the ordinary text-matching you're probably used to.

You'll also find the features that made BRIEF famous. Including our near-miraculous UNDO, that doesn't just un-delete your last keystroke. It actually reverses the effect of any command. So you can explore, experiment, even make gigantic mistakes with hideous consequences, and just "undo" them. And then continue to undo, backing-up through 300 steps.

What you want is what you get.

What you'll also find is an editor that delights in adapting itself to you. With the simple, menu-driven SETUP program, you can quickly customize indenting style, support for any of 36 compilers, keyboard configuration, rate of keyboard auto-repeat, and much more.

You can even edit BRIEF's macros to suit your taste. Or write your own in a snap.

Computer Language said that BRIEF is capable of "almost complete customization." And with new release 3.0, BRIEF is even more powerful and flexible.

For every programmer, it's a different BRIEF.

At last count, there were over 52,000 copies of BRIEF in the field, making life easier and working time more productive for all kinds of programmers—working in all kinds of languages. Programmers as different as Wayne Ratliff is from George Brandt. Which means there are over 52,000 different ways of working, and BRIEF fits them all.

Why not get BRIEF going to work for you? There's no risk. In fact, if you're not completely satisfied that BRIEF is exactly the editor you've been looking for, just return it within 30 days, and we'll return your money. No questions asked.

New BRIEF 3.0 highlights

- With our new C-like macro language, CBRIEF, C programmers can write macros on the fly and not have to switch between C and a different macro language.
- The new source level debugger works with both the original BRIEF syntax and the new CBRIEF macro language.
- Maximum lines per file has been increased to 4 billion.
- We've added to and improved template editing and smart indenting for ADA, Cobol, BASIC, FORTRAN, Modula-2, Pascal, and C.
- We've added editable, multiple keystroke macros that can be saved and restored.
- You can even save and restore your window set-ups.

Plus the features that made BRIEF famous.

- 300-level UNDO safety net
- Flexible windowing, unlimited quantity
- Unlimited number of files
- File size limited only by disk space
- Regular expression search and replace
- BRIEF's LISP-like macro language
- Toll-free tech support

BRIEF 3.0

The Programmer's Editor

Solution Systems™
541 Main St., Suite 410
South Weymouth, MA 02190
FAX 617-337-7719

Try BRIEF at our risk. $199 plus $6 shipping (U.S. and Canada). 30 day money back guarantee.

Name _____
Company _____
Address _____
City _____ State _____ Zip _____
Phone _____

Paying by ☐ Check ☐ MC ☐ VISA
Card # _____ Exp. date _____
Signature _____
Disk size ☐ 5-1/4" ☐ 3-1/2" Format ☐ DOS ☐ OS/2

Mail this coupon or call toll-free today.

800-821-2492
Extension 496
Massachusetts & outside U.S. call 617-337-6963

Circle 47 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Vaughn Vernon

Writing applications that travel

A resource string manager (with a little baggage)

I like to travel. I have been to Europe several times and once to Brazil. This winter I will be going to South America again—a whole month away from "Programmer Central." Yeeessss! While I have also been to the major regions of the U.S., there is nothing quite like traveling abroad.

Travel is easy these days. In addition, whenever we want, we can call across the country or overseas, or FAX information to almost any place on earth at any time of day or night. The point is, we are living in a very small world, one that is shrinking all the time.

As the world gets smaller, the demand for international software gets larger. However, there is relatively little written for international markets. Most companies are unaware of the potential in foreign markets, and those that are aware are unwilling to attempt to support products in that marketplace. Why? Probably because their software, from the beginning, had a serious design flaw. Its designers made the assumption that because "I" speak English, everyone else must speak English, too. Or perhaps the thought had never even ricocheted off the ol' grey-matter.

Those who had thought the problem through may have considered it too big a hassle to bother with foreign language translation: "Besides, 'my' product is really great. If 'they' want to use it, 'they' will just have to learn English." That is definitely the prevailing attitude among Americans. Sad to say, it becomes manifest in most American tourists. It goes something like this: "You idiot! Can't you hear? I said I want a hot dog! Honey, I told you we should have gone to Disney!" (Those who tour the US from abroad are rarely as demanding. They often make a great effort to stumble through our language, with some doing quite well.)

Okay, so you're not one of "those" guys. You think you'd like to be the neighborly type. Great—and that also means that you could sell your software to thousands or tens of thousands more people. After all, that's what makes the world go round.

Vaughn Vernon is author of Formation Desktop, Curses/PC, and Curses/VMS. Formation Desktop provides a GUI-like desktop interface with window, resource, and style management for character-mode programs. Vaughn can be reached at Aspen Scientific Corp., PO Box 72, Wheat Ridge, CO 80034-0072, USA or via e-mail at uunet!aspensci!vlv.

AVOID DEPENDENCIES

The following is the first serious mistake programmers who want foreign PCs to run their software could make:

```
printf("Hello, world\n");
```

You have embedded human language dependencies into your source code. This is just as bad a mistake for human language portability as hard-coding peeks and pokes is for operating system portability.

Resources, on the other hand, help an application stay away from human language dependencies. A resource is a program property which is not tied directly to the application through its source code. A resource could be a menu, a gadget such as a push button, a whole dialog box, or a simple string. GUI systems like Windows, PM, and X encourage heavy use of resources. Even OS/2 has a built-in message interface which uses string resources to hide human language dependencies. Any message that you see under OS/2, from the directory listing header to "Device not ready...," is supported via the message resource interface.

Resources are kept in an external file or files. Sometimes, as in the case of Windows and PM, resources occupy a special segment in the EXE file. But regardless of the file in which the resources are kept, strings are not embedded in the source code, and they will be available on demand.

BE RESOURCEFUL

The problem which arises in presenting a resource interface of any kind to developers is that you must be able to use it with your existing window tool. If I were to develop a full-blown resource manager that supported menus, dialog boxes and individual gadgets like push buttons and text edits, my resource manager would not know how

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

to interface with the myriad of window tools on the market. The real problem is not **how** your window tool works, it's **where** it gets its strings: from your source code, or from a program-independent resource file. If all of your strings—messages, window titles, input prompt titles, and menu strings—come from a program-independent resource interface, your programs will have the potential to run on a European, Latin American or any other "roman-based" desktop.

A STRING RESOURCE INTERFACE

I have developed a simple StringResource interface and a utility program which will work with any window tool kit. The basic idea is that you have an external ASCII text file, created with your program source editor, which contains the resource strings, and a simple library interface to load and retrieve the strings. The format of the resource file is shown below:

```
# strings.rsc  test file for resource strings
TEST_STRING1=100, "This is a string"
TEST_STRING2=101, "A string with %d arguments(s)"
TEST_STRING3=102, "This has an embedded %s"
```

This file has some basic rules for proper usage. First, all characters which follow the '#' are considered comments. Any characters which follow a '\' are taken as that character literally (as in C). Therefore, if you precede a '#' with a '\', the '#' will no longer be taken as a comment. You can also "escape" ('\') double quote characters which normally would represent a string beginning or end.

Each string resource is defined on a separate line. The first component of a string resource is its name and ID, followed by a comma. The name must be one that is acceptable to the C preprocessor. It will be used in your C program to identify the string. The name is followed by an equal sign and then a number. The number is the actual ID value used by the string resource interface and your application, and must be unique. Both the name and the ID become **#define** statements in a C header file. The ID number is followed by a comma, and then the string resource in double quotes. The string may contain embedded format characters which are supported by **printf**, etc. These are handy if you need to include run-time values or strings in a message.

RESOURCE IDS WITH MAKEID

The utility MAKEID (see Listing 1) has been provided to make it easier to generate the necessary resource ID names and values that refer to individual resources. Just run MAKEID, giving it as arguments the names of the string resource file or files you want resource IDs generated for. MAKEID places output onto stdout (standard output, the console). If you want the output directed to a C header file, just use the DOS redirection facility:

```
C> makeid strings.rsc >  myheader.h
```

If you want to append to an existing file, use the following append redirection facility:

```
C> makeid some.rsc >> myheader.h
```

MAKEID generates **#define** statements from the string name and ID numbers.

To use a StringResource file in a C program, you must include the file RESOURCE.H (Listing 2) and any header files generated by MAKEID. The RESOURCE.H header implements a simple linked-list interface which contains the resource names, IDs, and strings of a given resource file. It also provides the function prototypes and macros used in an application.

To load a string resource file into memory, you call **LoadResources**. This function takes a path to the resource file to be loaded, and returns a pointer to the StringResource list:

```
#include <stdio>.h
#include <resource.h>
#include <resrcid.h>
StringResource *resources;
resources = LoadResources("strings.rsc");
```

If a NULL pointer is returned, the resource file was not found or an error was detected.

When you have finished using the resources, you can free them from memory by calling the function **FreeResources**. You then pass the same pointer which was returned from **LoadResources**:

```
FreeResources(resources);
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Listing 1

```
/* makeid.c   creates a list of #define statements on stdout.
 *            can be redirected to a header file:
 *            C> makeid strings.rsc > myincl.h
 * (c) Copyright 1990 Aspen Scientific  All Rights Reserved */

#include <stdio.h>
#include "resource.h"

void fatal(char *str);

void main(int argc, char **argv) {
        register int    i;
        StringResource  *res;

        if (argc == 1) {
                fprintf(stderr, "usage:: makeid ResourceFile1");
                fprintf(stderr, " <file2> <file3> ..\n");
                exit(1);
        }
        for (i=1; i < argc; ++i) {
                res = LoadResources(argv[i]);

                if (res == NULL)
                        fatal("error loading strings.rsc");
                while (res) {
                        printf("#define\t%s\t\t%d\n",
                                GetResourceName(res), GetResourceID(res));
#ifdef DEBUG
                        printf("\t\"%s\"\n", GetResourceString(res));
#endif
                        res = NextResource(res);
                }
                FreeResources(res);
        }
        exit(0);
}

void fatal(char *str) {
        fprintf(stderr, "fatal: %s\n", str);
        exit(2);
}
```

Listing 2

```
/* resource.h   support for string Resource interface. */

static const unsigned   _ResourceMagic=12345;

typedef struct  _str_resource
{       unsigned char       *idName;  /* pointer to name string */
        unsigned char       *string;  /* pointer to resource string */
        unsigned            value;    /* integer id of value string */
        unsigned            magic;    /* make use resource is cool */
        struct _str_resource *next;   /* next resource */
} StringResource;

extern  StringResource *LoadResources(char *fileName);
extern  void           FreeResources(StringResource *res);
extern  StringResource *LookUpResource(StringResource *res, unsigned id);

#define GetResourceString(res)  \
        ((res)->magic==_ResourceMagic ? ((res)->string): \
                                        (unsigned char *)0)
#define GetResourceName(res)    \
        ((res)->magic==_ResourceMagic ? ((res)->idName): \
                                        (unsigned char *)0)
#define GetResourceID(res)      \
        ((res)->magic==_ResourceMagic ? ((res)->value):0)
#define NextResource(res)       \
        ((res)->magic==_ResourceMagic ? ((res)->next): \
                                        (StringResource *)0)
```

Listing 3

```
/* testrsc.c  test loading, looking-up and printing resource strings. */

#include <stdio.h>
#include "testrsc.h"
#include "resources.h"

void main()
{
        StringResource  *res = LoadResources("strings.rsc");
        StringResource  *current;

        if (res == NULL)
                fatal("error loading strings.rsc");
        current = LookUpResource(res, TEST_STRING1);
        printf("TEST_STRING1 = %s\n", GetResourceString(current));

        current = LookUpResource(res, TEST_STRING2);
        printf("TEST_STRING2 = ");
        printf(GetResourceString(current), 1);
        printf("\n");

        current = LookUpResource(res, TEST_STRING3);
        printf("TEST_STRING3 = ");
        printf(GetResourceString(current), "string\n");

        exit(0);
}

void fatal(char *str)
{
        fprintf(stderr, "fatal: %s\n", str);
        exit(0);
}
```

Listing 4

```
/* resource.c   support routines for string resources */

#include <stdio.h>
#include <string.h>
#ifdef __TURBOC__
#include <alloc.h>
#else
#include <malloc.h>
#endif

#include "resource.h"

static int              lineEatSpace(unsigned char *fbuf);
static StringResource * makeResource(unsigned char *fbuf);
static void             freeResource(StringResource *res);

/* LoadResources() loads a resource file into memory.
 * returns a resource list pointer, or NULL if an error is found. */

StringResource * LoadResources(char *fileName) {
        FILE            *fp = fopen(fileName, "r");
        unsigned char   fbuf[BUFSIZ];
        StringResource  *res = NULL, *cur, *tmp;

        if (fp == NULL) return (NULL);

        while (fgets(fbuf, sizeof (fbuf), fp)) {
                if (lineEatSpace(fbuf) == 0)
                        continue;
                tmp = makeResource(fbuf);

                if (tmp == NULL) {
                        FreeResources(res);
                        return (NULL);
                }
                if (res == NULL) res = cur = tmp;
                else {
                        cur->next = tmp;
                        cur = tmp;
                }
        }
        fclose(fp);
        return (res);
}

/* makeResource() creates a resource link on the free store.
 * returns a pointer to the new link. */

StringResource *makeResource(unsigned char *fbuf) {
        StringResource  *res;
        unsigned char   *idName, *idValue, *string, *end;

        res = (StringResource *) malloc(sizeof (StringResource));

        if (res == NULL) return (NULL);

        res->idName = res->string = NULL;
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

In order to get a pointer to an individual StringResource, call the function **LookUpResource**. You pass the StringResource pointer returned from **LoadResources** and the ID name of the string you want to load; that is, the ID name generated by MAKEID. Returned is a pointer to the StringResource item, or a NULL if it was not found:

```
#include <stdio.h>
#include <resource.h>
#include <resrcid.h>
StringResource *resources, *aStr;
unsigned char  *str;
resources = LoadResources("strings.rsc");
/* ... */
aStr = LookUpResource(resources, TEST_STRING1);
str = GetResourceString(aStr);
```

The function **GetResourceString** gives you the actual character string pointer associated with the resource. As its argument, it takes the StringResource pointer that is returned from **LookUpResource**.

If the string you are using requires run-time values to be provided, you can use the function **sprintf**. In this case, the StringResource string becomes the format string to **sprintf**, as shown in the following:

```
#include <stdio.h>
#include <resource.h>
#include <resrcid.h>
unsigned char   buffer[BUFSIZ];
StringResource  *resource, *str;
/* ... */
str = LookUpResource(resource, TEST_STRING1);
sprintf(buffer, GetResourceString(str), 1);
WindowWrite(0, 0, buffer);
```

The output would appear as:

```
This is a string with 1 argument(s)
```

The file TESTRSC.C (Listing 3) demonstrates how to use the StringResource interface. The file RESOURCE.C (Listing 4) implements the StringResource linked-list with its strings and ID names and numbers.

SOME BAD NEWS...

The bad news is that strings are not the only thing standing in the way of human language portability. There are issues such as date and time formats (which comes first, the day or the month?), and the type of decimal separator (a comma or a period?). However, these can be worked around with some thought, and there are some window tools that will do part of the work for you. The best advice I have for that problem is to know your users.

THE GOOD NEWS

You will be glad to know that this interface is portable not only to other human languages, but also to Unix and OS/2, and to any other ASCII-based O/S. The simple nature of the toolkit with ASCII text files and an elementary linked list almost guarantee its portability to a range of other platforms. All you really need is K&R- or ANSI-C with **malloc** and **free**.

Like traveling, writing code that will readily go from place to place requires some planning and work. With the right amount of forethought, however, both can be very rewarding.

"If I could put together a personal suite of programmer's utilities, I would include the C Programmer's Toolbox for a nifty Swiss-Army-Knife collection of utilities."
Warren Keuffel, *Computer Language*, June 1990

The C Programmer's Toolbox™ contains more than 20 tools for the serious C programmer. The tools are available in three different development environments: PC/MS DOS; Macintosh's MPW; and Sun's UNIX. Some of the major tools include:

CDecl	Translates C declaration statements to/from English *(helps you decipher those complicated C variable declarations and construct new ones). {MPW & Sun only}*
CFlow™	Determines how a program is organized and what is in the runtime library *(quickly shows you function hierarchy and interdependancies)*
CHilite™	Highlights and prints your C source code *(great for creating your final documentation). {MPW only}*
CLint™	Syntactically checks one or more of your C source code files & **generates ANSI function prototypes** *(use this to find those latent programming bugs)*
Cpp	Preprocesses C source code files that have ANSI macro definitions *(helps you port ANSI C code to/from your target environment). {MPW & Sun only}*
CPrint™	Reformats/beautifies your C source code *(makes code easy to read and ends disputes on source code formats).*
CritPath™	Determine a program's critical path. *(helps you improve your program's performance) {PC DOS only}*
CXref™	Cross references and checks usage of up to 17 different classes of symbols in your C source code *(cross references any size program and number of listings)*
PMon™	Monitors a program's execution *(non-invasively monitors function and system service execution. {PC DOS only}*

"I recommend getting both (Volumes)... compared to UNIX tools, the Toolbox tools have more options and provide more detailed information..." —*Kenji Hino, The C User's Journal, February 1990*

C Programmer's Toolbox™ / MPW (Macintosh) / PC (PC/MS DOS) / Sun (UNIX)

New 2.1 Release

The tools include support for ANSI C, Microsoft C *(new support for 6.0)*, Turbo C, Sun UNIX C, Macintosh's MPW C, THINK C *(new support for 4.0 OOPs)* and more. The Toolbox is *fast*, handles *any size program* and is easy to use. Several tools also include a C preprocessor that accepts K&R, ANSI and other C dialects. Improve your productivity and your program's performance and quality today:

C Programmer's Toolbox™
MPW ($295), PC Volume I or II ($99.95),
PC Volume I & II ($175), Sun ($495)
C Programmer's McTool Series™ (Macintosh - non-MPW)
McCPrint™ ($99.95), McCLint ($149.95)

"The documentation is exemplary... strong internal design..." —*Will Watts, I.XE, April 1990*

Call and Order today...
(CA residents: please add sales tax; Overseas: contact us for details)

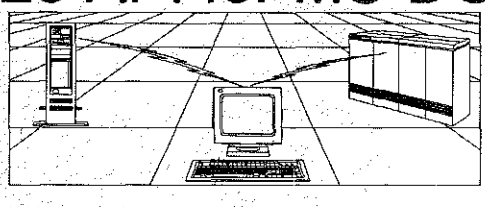
MasterCard VISA

MMC AD Systems™
Box 360845
Milpitas, CA 95036
Phone: (415) 770-0858
Fax: (415) 770-0116

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        idName = fbuf;  /* parse the line... */
        idValue = strchr(fbuf, '=');

        if (idName == NULL || idValue == NULL) {
                freeResource(res);
                return (NULL);
        }
        *idValue++ = '\0';

        string = strchr(idValue, ',');

        if (string == NULL) {
                freeResource(res);
                return (NULL);
        }
        *string++ = '\0';
        if (*string != '\"') {
                freeResource(res);
                return (NULL);
        }
        ++string;
        end = strrchr(string, '\"');

        if (end == NULL) {
                freeResource(res);
                return (NULL);
        }
        *end = '\0';

        /* set up the link ... */
        res->idName = (unsigned char *) malloc(strlen(idName) + 1);
        res->string = (unsigned char *) malloc(strlen(string) + 1);

        if (res->idName == NULL || res->string == NULL) {
                freeResource(res);
                return (NULL);
        }
        strcpy(res->idName, idName);
        strcpy(res->string, string);

        res->value = (unsigned) atoi(idValue);
        res->magic = _ResourceMagic;
        res->next  = NULL;

        return (res);
}

/* lineEatSpace() is your basic text line parser.
 * returns number of "useable" characters; 0 if all white or comments. */

static int lineEatSpace(unsigned char *fbuf) {
        register int    quote=0;
        unsigned char   tmp[BUFSIZ+1];
        unsigned char   *p1 = fbuf, *p2 = tmp;

        memset(tmp, 0, sizeof (tmp));

        while (*p1 && *p1 != '\n') {
                switch (*p1)
                {
                case    '#':  *p1 = '\0';  break;
                case    '\t': ++p1;  continue;
                case    ' ':  if (!quote) {
                                        ++p1;
                                        continue;
                              }
                              break;
                case    '\\': ++p1;  break;
                case    '\'': quote = (quote == 1 ? 0: quote == 2 ? 2:1);
                              break;
                case    '\"': quote = (quote == 2 ? 0: quote == 1 ? 1:2);
                              break;
                default:      break;
                }
                if (*p1 == '\0') break;
                *p2++ = *p1++;
        }
        memcpy(fbuf, tmp, BUFSIZ);
        return (strlen(fbuf));
}

/* FreeResources() frees an entire resource list */

void FreeResources(StringResource *res) {
        StringResource  *tmp;
        while (res) {
                tmp = res->next;
                freeResource(res);
                res = tmp;
        }
}

/* freeResource() frees an individual resource */

void freeResource(StringResource *res)
{
        if (res == NULL) return;

        if (res->idName)  free( res->idName );
        if (res->string)  free( res->string );
        res->magic = 0; /* no reuse */
        free( res );
}

/* LookUpResource() finds a resource matching the given ID.
 * returns the resource pointer, or NULL if not found. */

StringResource *LookUpResource(StringResource *res, unsigned id) {
        while (res && res->magic == _ResourceMagic) {
                if (res->value == id) break;
                res = res->next;
        }
        return (res && res->magic == _ResourceMagic ? res:NULL);
}
```


X.25 API for MS-DOS



Simultaneous Multi Session Communications

100% Error Free Protocol

Speeds from 2400bps to 64Kbps

Loads either as TSR or dynamically from Application

Provides Language Independent "8086 Software INT" Application Interface

Operates with Hayes AUTOSYNC compatible or UDS Sync-Up Modems

X25API Development Kit Includes:

- Programming Guide
- Technical Reference
- X25API Object Code
- C Language Demo Program

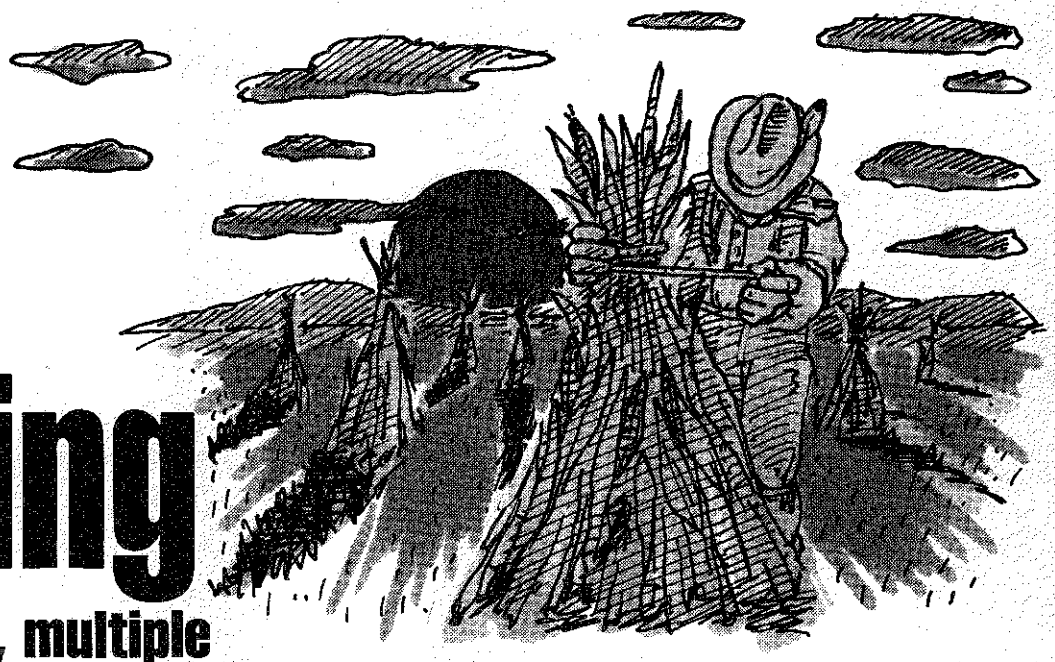
Microdyne Corporation, PC Technologies Division
491 Oak Road, Ocala, Florida 32672
904/687-4633, FAX 904/687-3392

PJ CODE All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 16 and 80.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By Brett Glass

Late binding



Polymorphism, multiple inheritance, and less complex code

Object-oriented languages owe a lot of their power and flexibility to *late binding*—the ability to decide at runtime what method to use on a particular object. When you write a call to an ordinary procedure or function in a non-object-oriented language such as C or Pascal, the compiler knows for sure at compile time what routine it's going to call. In an object-oriented language, however, the code that's executed may depend on the class of the object on which a method has been invoked—an issue that can't always be decided until the code actually runs.

WHAT IS BINDING?

Binding is the process by which a name (an identifier in a programming language) is associated with an object (such as a variable or a subroutine). Most of us who use compiled programming languages are inclined to think of binding as something which happens during compilation and linking, and then never again: once a function or variable has a name, that name belongs to it forever (subject to the language's rules about the lifetime and scope of variables). Almost all compiled, non-object-oriented languages do, in fact, work this way. They're said to have *early binding*, because the naming is all done before the program runs.

By contrast, many languages that are interpreted or "comterpreted" (that is, compiled to an intermediate form which is then interpreted) allow definition (or redefinition) of a function at runtime. In LISP, for example, you define a function by calling another function called **defun**. The LISP expression

```
(defun increase (a) (plus a 1))
```

defines a function **increase** which adds one to its argument, *a*, and returns the result.

Because **defun** is a bona fide LISP function, it can be executed from within another function. In other words, if the function **increase** were already defined, and your program evaluated the expression:

```
(defun increase (a) (plus a 25))
```

the definition of the function would instantly change, and the next invocation would use the new definition. A function can even modify its own definition safely; the "old" version of the code continues to execute until the function exits.

In the Smalltalk language, it's possible to make instant modifications to the definitions of data structures as well as to functions or methods. On Xerox's Dorado, the machine on which much of the original Smalltalk development was done, one could literally change the operating system on the fly while it was running! Object-oriented versions of LISP with "flavors" have similar properties. When the definition of a class in one of these languages is changed, the newly-defined class then becomes a subclass of the old one, inheriting everything that was not changed. The machine keeps running, even if objects of that class were part of the operating system. Eventually, once all objects of the old class are destroyed, the class definition can finally be removed from the system.

Such situations are the quintessence of late binding, in which a function or method is (for all practical purposes) looked up by name each time it's invoked. Most compiled languages, however, take a middle ground; they don't let your program create new methods at runtime, but they do allow it to choose one of several different methods that already exist.

POLYMORPHISM

To understand why late binding can be useful, you need to understand another trait of object-oriented programming languages: *polymorphism*. Suppose you are writing a graphics package which lets you draw points, circles, rectangles, and other objects on a display. Like a good object-oriented programmer, you begin your work by defining a base class for graphics objects, which might look like this:

Brett Glass is a free-lance writer, programmer, and consultant residing in Palo Alto, California. He can be reached at rogue@well.sf.ca.us on Internet, on CompuServe at 72267,3673, or as glass on Bix.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C++:

```
class GraphObj {
  int x,y; /* Location of object */
public:
  int getx {return x;}; /* Use inline functions
                           in C++ */
  int gety {return y;};
  void moveto (int newx, int newy)
       {x = newx; y = newy;};
  };
```

Turbo Pascal:

```
type GraphObj = object
    x,y : Integer; {Location of object}
    function getx : Integer;
    function gety : Integer;
    procedure moveto (newx,newy : Integer);
  end;
```

Notice the similarities here between the C++ and Turbo Pascal approaches. In both, our base class gives information about the location of the object and provides access methods for the *instance variables*—that is, the data fields of the record that represent the object. (The terminology differs between the languages: C++ calls a method a *member function*, and you usually don't hear the term "instance variable" outside of Smalltalk circles.)

Image

Database Processing
Storage Management

Image is an innovative C library with an unprecedented combination of relational power, efficient memory management, and flexible C-language control.

You gain power! *Image* operations treat entire data sets as units. Sort, search, select or update any number of records with a single call.

You retain control! *Image* always allows direct access to your data. Any C operation, any record, any time.

You get started quickly! *Image* is easy to use, with plenty of sample programs to help you get started.

You finish sooner! *Image* programs usually work the first time. *Image* gives database power with elegant simplicity. Complex problems become simpler.

Your code is efficient! *Image* gives you an integrated family of fast, tight, byte-oriented operations.

You save time and space! *Image* keeps sets in expanded memory or RAM, spilling to disk automatically, and only as needed.

No Royalties Satisfaction Guaranteed

Venn Set Processing Software
P.O. Box 1008
Ann Arbor, MI 48106

Only $129.95

☐ 5.25 ☐ 3.5 Density ____
☐ Microsoft ☐ Turbo ☐ Zortech
☐ Visa ☐ M/C ☐ COD

NAME ____
ADDRESS ____
CITY ____
STATE ____ ZIP ____

____
Exp ____

ORDER TODAY 1-800-628-8369

Circle 29 on Reader Service Card

In either Turbo Pascal or C++, you could go on to derive the classes Point, Circle, and Rectangle, which would inherit the instance variables *x* and *y* and add other fields and methods germane to that shape. Here are some definitions that might be used for the Point subclass:

C++:

```
class Point : GraphObj {
  int color;
public:
  void setcolor(int newcolor);
  int getcolor();
  void draw();
  };
```

Turbo Pascal:

```
type
  Point = object (GraphObj)
    color : Integer;
    procedure setcolor(newcolor : Integer);
    function getcolor : Integer;
    procedure draw;
    end;
```

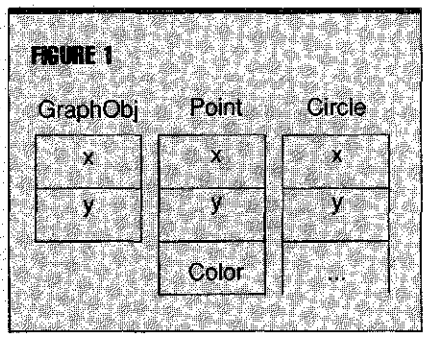
Note that the access functions **getx** and **gety** will also work on members of this derived class (and all other classes derived from GraphObj), just as they do on the base class. It's easy to imagine how the compiler can make this possible: because each member of a derived class has all the fields of the base class, plus (possibly) others, the internal data structure that represents the derived class is just an extension of the structure used for the base class. This means that the offsets of the fields they have in common will be the same.

Therefore, if the functions **getx** and **gety** simply retrieve the values using their offsets from the beginning of the record (a convenient operation on nearly all microprocessors), they are guaranteed to work on any derived class. This trait is called *polymorphism*, which means, literally, "the property of having many shapes."

This simplest type of polymorphism (as seen in Figure 1), which makes use of the way data structures are extended as the class hierarchy grows, does not require late binding. A method defined in the base class can work on a descendant regardless of what structures have been added on. But what if, due to the logistics of the situation, structure *does* matter?

In this case, it would certainly be helpful if the compiler were able to hide the differences between the derived types, and to allow the use of the same method name to perform an identical

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



operation. This more advanced form of polymorphism is achieved by using late binding.

DRAW THYSELF!

Let's return to our graphics example to see how this works. Since each of the classes derived from GraphObj is something that can be drawn on the screen, each will presumably have a method called **draw**. For the class Point, for instance, it might look like this:

```
C++

void Point::Draw()
{
  Plot(getx(),gety());
}

Turbo Pascal:

procedure Point.Draw;
  begin
  Plot(getx,gety);
  end;
```

So if the variable **thing** represented a graphics object, and each of the derived classes implemented a method called **draw**, one could safely say:

```
thing.draw(); /*C++*/  or
thing.draw; {Turbo Pascal}
```

and the object would be drawn. So long as the class of **thing** is known at runtime, the compiler knows which draw routine to use and can perform early binding on the name **draw**.

Now, suppose you are using a "display list" approach to drawing your screen. You want to run down a list of objects of different kinds and draw them all, using the proper method for each. But how to determine which method to use? One way to do this is to build a tag field into each node of the list indicating the type of the object, and use a case or switch statement in the loop that does the traversal.

There's a more efficient way, however. Both C++ and Turbo Pascal have

The Heap Expander™ v 2.05

Gain virtually unlimited heap space.

Function

Provides interface routines so that your programs use expanded memory, extended memory or conventional memory. As memory becomes full, The Heap Expander swaps to disk.

Utility

The Heap Expander comes with complete source code, libraries and documentation plus four utility programs that simplify development of expanded memory programs.

Guarantee

Guaranteed by The Tool Makers to perform as specified with an unlimited money-back refund.

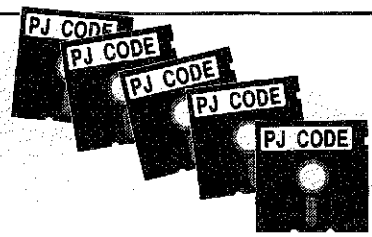
Includes libraries and source code for all current versions of Turbo C and Microsoft C.
System requirements: IBM PC, XT, AT or clone compatible with MS-DOS or PC-DOS v. 2.0 or above.

MC/VISA/COD
800-248-1045 ext 100
USA
800-952-5560 ext 100
Idaho

still just **$79.95**

THE TOOL MAKERS
P.O. Box 2151
Santa Cruz, CA 95063
408-458-0690

Circle 166 on Reader Service Card



On disk from:
Programmer's Journal
P. O. Box 70167, Eugene, OR 97401-0110
Call toll free: 800-234-0386
In Oregon call: 503-747-0800

All Code Disks $10.00 each.
To order, see ad on page 80.

PJ Code is available on the following bulletin board services:

Buffer Board
538 Cardinal Way
Springfield, OR 97477
(Data) 503-747-7636
The Buffer Board contains all of PJ's code from Sept/Oct 1987 issue through the current issue.

Generation Five Inc.
12061 Tech Road
Silver Spring, MD 20904-9998
(Voice) 301-680-3600
(Data) 301-680-9300
Accessible via PC Pursuit, Washington, D.C.
From Files submenu, select "P" for Personal Downloads.

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Ithaca Street BBS
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

PCS-49 on CompuServe
Section 10
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 703-450-4181

Programmers Corner
5430F Lynx Ln. #224
Columbia, MD 21044
213-374-3478
(Voice) 301-740-5880
(Data) 301-596-1180
(Data) 301-997-3744

The Silicon Zone
820 Tivoli Circle #203
Deerfield Beach, FL 33441
(Voice) 305-360-9362
(Data) 305-360-9376

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Confusing Code?

```
#include <stdio.h>
text_count(){int c,nlines,nwords,nchars,inword;
inword=NO;nlines=nwords=nchars=0;while((c
=getchar())!=EOF){++nchars;if (c=='\n')
++nlines;if ((c==' ')||(c=='\n'))inword=NO;else
if(inword==NO){inword=YES;++nwords;}}printf(
"%d %d %d\n",nlines,nwords,nchars);}
```

C It Your Way!

```
#include <stdio.h>

text_count ()
{
    int c, nlines, nwords, nchars, inword;

    inword = NO;
    nlines = nwords = nchars = 0;
    while ((c = getchar()) != EOF) {
        ++nchars;
        if (c == '\n')
            ++nlines;
        if ((c == ' ') || (c == '\n'))
            inword = NO;
        else if (inword == NO) {
            inword = YES;
            ++nwords;
        }
    }
    printf ("%d %d %d\n", nlines, nwords, nchars);
}
```

NOW with **C-Clearly™**, format C source code exactly the way you want it. C-Clearly's context sensitive analysis will format any C program in your own personal or corporate style.

EASY to use, C-Clearly's style templates are a snap to modify, since they resemble C source code you edit into your preferred format. Templates are included for several common styles as well as standard K & R.

LISTINGS can also be created with function names and comments highlighted for improved readability. Listing options include line numbers, headers and/or footers and flow lines.

IDEAL for making obtuse code clear. Allows all of your source code to be presented in a consistent format of your choosing. Also great for code walkthroughs and final documentation listings.

WORKS with all IBM PC, XT, AT, PS/2 and compatibles, with 512K RAM. Automatically processes all include files and pre-processor statements. ANSI-C compatible. Not copy protected.

C-Clearly $129.95

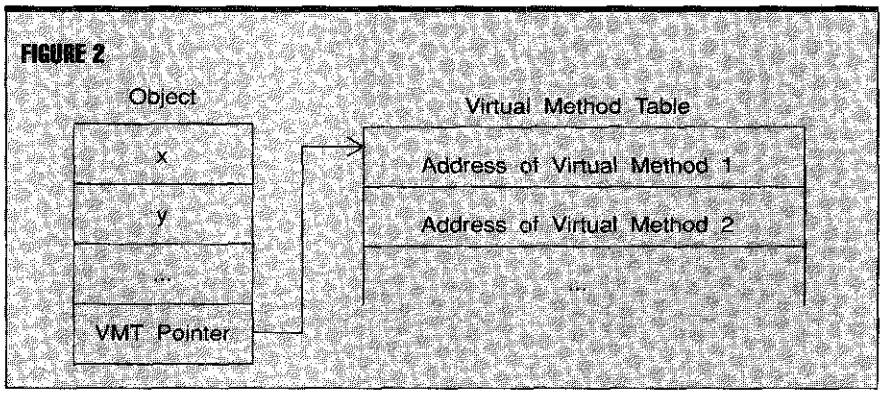
Shipping & Handling USA $5, Canada/Mexico $10; Other Countries $15; CA Residents add sales tax. Visa/MasterCard/COD accepted.

For orders or information call:

1-800-662-8266

V COMMUNICATIONS, INC.
4320 Stevens Creek Blvd., Ste 275, Dept. PJ6
San Jose, CA 95129 (408) 296-4224

Circle 341 on Reader Service Card



the ability to build information about which methods to use directly into the object.

A *virtual method* is a method selected at runtime via a pointer or tag built into the object on which the method is invoked. To use virtual methods in our graphics example, we must redefine the base class GraphObj in the following manner:

C++:

```
class GraphObj {
  int x,y; /* Location of object */
public:
/* Use inline functions in C++ */
  int getx {return x;};
  int gety {return y;};
  void moveto (int newx, int newy)
       {x = newx; y = newy;};
  virtual void draw; /* Virtual method to draw */
                     /*   each derived class   */
};
```

Turbo Pascal:

```
type
  GraphObj = object
    x,y : Integer; {Location of object}
    function getx : Integer;
    function gety : Integer;
    procedure moveto (newx,newy : Integer);
    { Virtual method to draw each derived class }
    procedure draw; virtual;
        end;
```

The second-to-last line of each definition—the one with the word "virtual" in it—indicates that every object of the class GraphObj, or of any derived class, will have something in it which will tell the program, at runtime, which routine is used to draw the object. Given this definition, one can write constructs such as:

C++:

```
GraphObj *p;
...
p = &anyGraphObj;
p->draw(); /* Same as (*p).draw; */
```

Turbo Pascal:

```
var p : ^Graphobj;
...
p := @anyGraphObj;
p^.draw
```

and everything will be drawn properly. The last statement in each of these examples becomes a generic request: it says, in effect, "Whatever you are, draw yourself!"

Here's what's going on behind the scenes. Each object contains within it a pointer which in turn points to a *virtual method table*—an array of vectors that point to the methods for that class. The compiler generates code to find the correct method and execute a call to it. This is shown in Figure 2.

In most language implementations, objects of the same class can share virtual method tables (VMTs) by having identical VMT pointers. But some—including many implementations of Apple's Object Pascal—put a VMT inside every object instead of referencing a common table with a pointer. This saves one dereferencing step, and affords more flexibility if one wants to change an individual object's methods by tweaking the table. The downside of this approach is that objects take up a lot more room.

MULTIPLE INHERITANCE

The situation becomes even more complex when, as in C++ 2.0, a language features multiple inheritance. In this case, a class must have multiple virtual method tables—one for each of its parent classes with virtual methods. To complicate matters even further, the semantics of type casts have to be modified in order to make a parent class's methods work on a child with multiple inheritance.

Here's why. Suppose, for example, we created a C++ class for a table that is to be displayed on the screen. We might call it ScreenTable, and make it a descendant of a class called Table (which

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

contained the "table-like" elements) and a class called GraphObj (which contained information on how to display it). An object of the class ScreenTable might look like Figure 3 when laid out in memory.

Now suppose we were traversing our display list, looking for where each object was on the screen. To do this, we'd invoke the method **getx** from the parent class GraphObj on each object:

```
GraphObj *p;
...
p = (GraphObj *)&tableObject;
/* Or an equivalent statement to
extract the address from a list node */
...
i = p->getx();
```

But wait! If the cast of &tableObject produced the address of the beginning of the object (as pointer casts usually do), the subsequent call would use offsets from that point, and would act on the wrong fields of the data structure. For this reason, it is the cast operation that actually produces the address of the GraphObj part of the ScreenTable object. This clever trick—and others like it—allow polymorphism to work even with multiple inheritance.

USE AS DIRECTED

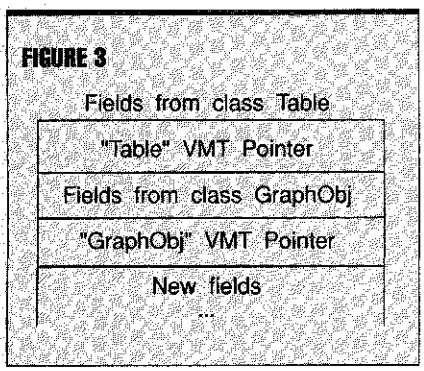
For the sake of efficiency, late binding should be used only when truly necessary. For best results, a compiler should not direct a call through a VMT unless it recognizes that the method to be used *cannot* be determined at compile time. Most good compilers use this optimization, although some of the older ones (and some C++ preprocessors) do not.

As a programmer, you should also be aware that late binding can increase a program's memory requirements. All of those pointers to VMTs—one in every object of a class with virtual methods—can take up a significant amount of space. The impact on speed is not quite as clear. A "double jump" through a pair of vectors may or may not be slower than a case statement; it depends on the number of choices and the architecture of your machine.

Late binding pays off, however, by making programmers' lives easier. It makes objects appear to "think for themselves" and "do the right thing"—part of the gestalt of object-oriented programming. It's also useful if you are interested in providing other programmers with extensible "toolboxes" of objects. Most of the source code packages available for Turbo Pascal and C++, for example, make extensive use of virtual methods.

Finally, and perhaps most importantly, late binding can prevent mistakes by letting the machine do the work and by reducing code complexity. In this day and age, when hardware is getting cheaper and programming errors more expensive, this is surely a consideration that no programmer can realistically afford to ignore. PJ

FIGURE 3



Turn your favorite C compiler into a powerful database manager with the
C/Database Toolchest
The C/Database Toolchest™ adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the C/Database Toolchest™, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.
The C/Database Toolchest™ includes three major components:
1) An advanced B+tree library gives you instant access to your data.
2) A high-level ISAM library provides you with an easy-to-use C interface, and
3) A complete database manager (with C source code included) shows you how to create impressive applications.
You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.
The C/Database Toolchest™ supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.
About the only thing that the C/Database Toolchest™ doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.
Now Only $19.95!
Order Coupon
Name
Street
City
State Zip
Telephone
Paying By Money Order Check
Visa MC AX Disc.
Card#
Exp. Date
Disk Size 5¼" 3½"
Qty. Product Price Subtotal
C/Database Toolchest $19.95
C/Database Library Source $10.00 B+tree & ISAM library source code
Add Shipping ($5 USA, $20 Foreign)
Texas Residents Add 8% Sales Tax
Total Amount of Your Order
Order now by calling our toll free number or mail the coupon to:
Mix Software
1132 Commerce Drive
Richardson, TX 75081
1-800-333-0330
60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001
MIX software
C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (create, read, & write MS-DOS file systems on non-MS-DOS computers)	$750
Cheaper!	CSource Application Program Generator by MBAC (includes *all* source code; generator & libraries)	$385
	GraphiC 5.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	Aspen PC Curses (System V compatible, extensive docs)	$290
NEW!	Time Series Management Tool (collection of routines for manipulating time series)	$250
	TurboTEX (Release 2.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	db_File & db_Retrieve Bundle by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
	Install 2.3 (automatic installation program; user-selected partial installation; CRC checking)	$195
Cheaper!	SilverComm "C" Async Library(complete asynchronous communications library)	$190
	Wendin-DOS Plus (self-bootable, multitasking, multiuser MS-DOS replacement; includes XTC editor)	$180
	QuickGeometry Library (large collection of mathematics, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC)	$170
NEW!	Rogue Wave tools.h++ Class Library (collections, B-trees, file space management, time& date, persistent objects; complete user's guide)	$165
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$165
	Essential resident C (TSRify C programs, DOS shared libraries)	$165
	TurboGeometry (library of routines for computational geometry, Version 3.0)	$160
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	Hold Everything (spawn new programs; swap parent to EMS or disk; handles video, interrupts, & environment; returns error level)	$150
Windows!	RED (program development editor for Microsoft Windows; keystroke recording & playback; unlimited undo)	$150
	C Generator (generates C code to read & write file records defined with C structure syntax)	$150
	Cephes Mathematical Library (over 100 high-quality, double-precision scientific functions)	$150
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
Cheaper!	CBTree (B+tree ISAM driver, multiple variable-length keys)	$135
NEW!	Envoy Communications Package (includes editor, script language, redefinable keys, help screens, menu-driven interface	$130
	C++ Object Library (virtual windows, I/O, lists, file free space management, keyed ISAM file I/O)	$125
	Rogue Wave math.h++ Class Library (C++ overloadings for standard operators, matrix inversion & FFT; complete user's guide)	$125
NEW!	Kier CountryLib (date, time, measurement, numeric, and simple language conversion; helps build country-independent software)	$120
NEW!	SilverWare "C" EMM Library (60 interface functions to EMM Version 4.0)	$120
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$115
	PowerSTOR (upto a gigabyte of heap space on extended memory, expanded memory, and/or hard disk)	$110
	B-Strings (dynamic string handling; cut, copy, paste, search, user input, etc.; non-fragmenting memory management)	$105
	Minix Operating System (Version 1.3; U••x-like operating system, includes manual; specify AT or XT)	each $105
	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, gateways)	$100
Cheaper!	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$90
	Disassembler (8086/80286/80386/80C186 with 80287/80387/80C187 NPX)	$90
	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$85
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
Updated!	Otter 2.0 (beautiful theorem-prover by Bill McCune; includes manual & two books by Wos; complete starter kit)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
	HY-PHEN-EX (a hyphenator for American English with over 4,800 rules)	$75
	eval() (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Symtab/Ptree (general-purpose symbol table/parse tree construction and management package; specify Symtab or Ptree)	each $75
	XSPAWN (swap parent process to expanded memory or disk before running child)	$70
	Heap Expander (virtual memory manager using expanded memory, extended memory, and disk space)	$65
	Foundations-1 C++ Class Library (ASCII record I/O, bit arrays, exception handling, B-tree, persistent objects)	$60
	Quincy (interactive C interpreter)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
	Async-Termio (Unix V compatible serial interface for MS-DOS; stty, ioctl, SIGINT, etc.)	$55
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Level 0 ISO standard Pascal with some Level 1 features)	each $50
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	CLIPS (rule-based expert system generator, Version 4.3; advanced manuals available at additional cost)	$50
Updated!	PCHRT (microsecond timers and delays, interrupt profiles, timer tick management)	$45
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$45
	BOLS Data Utilities (Sort/Merge/Select; five easy-to-use ASCII file management routines)	$40
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud)	$40
	FlexList (doubly-linked lists of arbitrary data with multiple access methods; specify C or C++)	each $40
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
NEW!	Classy Forms (screen forms in both C and C++ versions)	$40
	RXC & EGREP (Regular Expression Compiler and Pattern Matching; RXC makes finite state machine from regular expression)	$35
	NIH Class Library (collection of basic C++ classes by Keith Gorlen; V2.204 (V3.0 Pre-release); uses C++ R2.0)	$35
	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
NEW!	Spell Pack (5 spelling programs: Ispell, Microsp, Sp, Cspella, & the Dawg utilities)	$30
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
	6-Pack of Editors (baker's half-dozen public domain editors for use, study & hacking; includes microEmacs 3.10 & Stevie, a vi clone)	$30
Upgrade!	Crunch Pack (18 file compression & expansion programs)	$30
	ICON (string and list processing langauge, Version 8.0)	$25
	PC-MAIL (UUCP mailer by Wietse Z. Venema; send, receive, and manage UUCP mail)	$25
NEW!	PERL for MS-DOS (C, sed, awk, and shell all rolled into one language)	$25
Updated!	FLEX (fast lexical analyzer generator; new, improved LEX; official BSD Version 2.3 with docs)	$25
NEW!	Using C++ Library (the code from the book by Bruce Eckel and then some; Zortech 2.0 compatible)	$25
	VAL Linker by David Troendle (a complete linker; excellent for learning about linking)	$20
	A68 (68000 cross-assembler)	$20
	XLT Macro Processor (general purpose text translator)	$20
	Data	
	Moby Part-of-Speech (200,000 words and phrases described by prioritized part(s)-of-speech)	$120
	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$80
	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
	Dictionary Word List (234,932 words in alphabetical order)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	each $30
	Interactive Computer Ephemeris (high-precision moon, sun, planet & star positions; USNO (no source) & Downey 4.15 (C source))	$30

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3464 USA

Voice: (512) 258-0785
FAX: (512) 258-1342
E-mail: info@acw.com

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Circle 30 on Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By M. Steven Baker

Tools for 386 DOS

A look at some DOS extender development tools

In the last **386 NOW** article, I looked in some detail at FORTRAN compilers available under 386 DOS Extenders. Since this issue of *PJ* focuses on OOP and new languages, I thought a short overview of program development tools and available languages would be most appropriate.

Also, I've been stalling my look at the latest round of C compilers until the new symbolic debuggers that will enhance these compilers arrive. Watcom C/8.0 is the first vendor to deliver a 386 windowing debugger comparable to what many DOS programmers have become used to—Microsoft's CodeView and Borland's Turbo Debugger. MicroWay has a beta test version of a debugger for their 386/486 compilers, which I've recently started using. MetaWare is finishing a debugger and profiler for their 386 DOS compilers. SVS is working on a windowing version of SVSDEBUG, their command-line symbolic debugger. SAS has moved Lattice's 386 DOS and other C compiler development projects to SAS in North Carolina. A number of Lattice staff who chose not to relocate from Illinois are being let go, and it's not clear whether Lattice will finish their 386 C development efforts.

Several other developments in the PC market are shaping 386/486 development tools. Street prices for a 486 chip have been inflated—$1,200 to $1,600 over the last few months. This is more than the cost for an entire 33 MHz 80386 motherboard with 4 megabytes of RAM. The C1 step of the 80486 chip is now being sampled. This appears to be a clean mask and should allow Intel to ramp up production of the 486.

Weitek has funded a number of compiler vendors to add *instruction scheduling* to their code generators. Without instruction scheduling, the Weitek 4167 provides only a marginal improvement over the math coprocessor built into the 486. Cyrix now also offers a memory-mapped version of their 83D87 math chip. This reduces the communication bottleneck between the 386 and 387 and provides for much faster execution. By design, Intel limited the performance of the 387 by restricting 386/387 communication. This ensured that the 80486 would easily provide improved floating point performance over the 80386/387. A number of specific 486 optimizations are also available to speed code execution. So the 386 (and now 486) DOS development tool market is in considerable flux.

ASSEMBLY LANGUAGE

DOS assembly language tools for the 386 have traditionally been supplied by Phar Lap. Their 386|ASM/LINK package includes an assembler, a linker, a librarian, and a mini-debugger. The Phar Lap assembly language tools generate EASY-OMF object modules which are supported by most 386 compiler vendors. EASY-OMF extends various fields in the Intel 8086 object module to 32 bits in order to support the 386. EASY-OMF object modules are marked by a comment record at the beginning of the file containing the string "80386."

Microsoft's MASM 5.0+ and Borland's TASM can also be used to assemble 80386 code and generate a different (Microsoft extension) 386 object module format. Neither Microsoft nor Borland offers a linker or librarian to handle 386 object files. Ergo Computing bundles a 386 linker and librarian written by Lahey Computer Systems with their OS/386 DOS extender that can handle both Microsoft and Phar Lap 386 object modules. The Lahey librarian can also convert Phar Lap libraries to its own 386 Lahey library format. These tools are the same language utilities supplied with Lahey F77/LEM-32, their 386 FORTRAN development system.

MetaWare bundles a very useful binary dump utility (BD.EXE) with their High C-386 and Professional Pascal-386 which handles both Phar Lap and Microsoft 386 object modules. It will dump out both object module formats for inspection and allow patching an object module. BD's most valuable feature, however, is its ability to convert from

M. Steven Baker is the editor of Programmer's Journal *and a co-author of* Extending DOS, *published by Addison Wesley. He can be reached occasionally at the* PJ *offices with red pencil in hand.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

People are talking about us.

F77L-EM/32

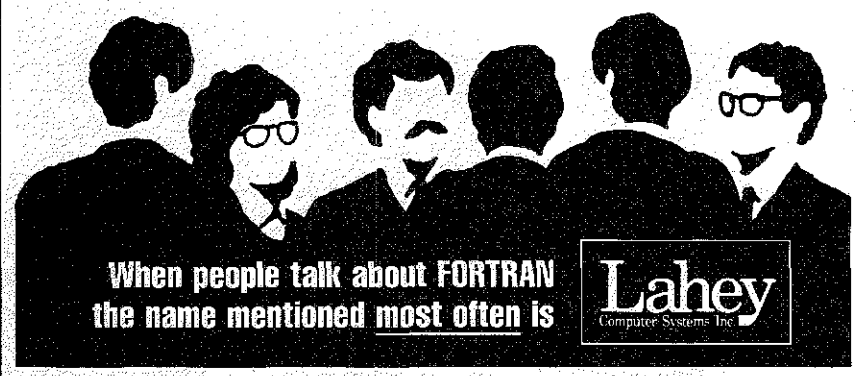
Port 4GB mainframe programs to 80386s with this 32-bit DOS-Extender compiler. The Winner of *PC Magazine's* 1988 Technical Excellence Award just got better. New Version 3.0 and OS include: Editor, Make Utility, Virtual Memory Support, DESQview Support, New Documentation and Free Unlimited Runtime Licenses. F77L-EM/32 $895 OS/386 $395

F77L

The compiler of choice among reviewers and professionals. Includes a Debugger, Editor, Profiler, Linker, Make Utility, Weitek and 386 Real-Mode Support, Graphics. $595

Lahey Personal Fortran 77

New Version 3.0: Full ANSI 77, Debugger, Editor, Linker, Library Manager, Microsoft and Borland C interfaces, 400 page Manual, Unbeatable Price. $99



Contact us to discuss our products and your needs. **(800) 548-4778**
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256

FORTRAN IS OUR FORTE

Circle 44 on Reader Service Card

C Programmers

Explore a world of DOS memory

Beyond 640K

Use PowerSTOR to exploit:
- LIM 4.0 or LIM 3.2 Expanded Memory
- AT Style Extended Memory
- Hard Disk Space
- Any Combination

- PowerSTOR is the easy way to use system memory resources. It self-configures at run time, under control of your software.
- PowerSTOR makes system memory resources available to your software as STORheaps. STORheaps can be allocated and de-allocated dynamically. Up to a gigabyte of PowerSTOR memory can be used as one huge STORheap or many smaller STORheaps.
- PowerSTOR uses only 15K in the lower 640K.
- PowerSTOR with K&R standard source code – only $92.50.

Call Toll Free in the US and Canada

1-800-621-5663

Acme Software

P. O. Box 40734, Portland, Oregon 97240 — Phone (503) 235-2816

Circle 305 on Reader Service Card

one object module format to the other. This BD utility can be purchased separately, if desired.

HIGH-LEVEL LANGUAGES

A number of high-level languages are available for developing 386 DOS extender applications. Compilers targeted for the 386 include Ada, APL, BASIC, C, C++, Cobol, Forth, FORTRAN, Lisp, Pascal, PL/I, Prolog, and Smalltalk languages. C and Pascal compilers from MetaWare for the 386 DOS extender environment were the earliest products available, appearing in 1987. In fact, 386 DOS extender development depended on the availability of MetaWare High C-386. High C-386 was initially used for generating portions of the runtimes, utilities, assemblers, linkers, librarians, and debuggers.

MetaWare compilers were followed by the introduction of 386 compilers, which had been retargeted from the minicomputer and Unix workstation world. These products include compilers from Silicon Valley Software (SVS), MicroWay (retargeting Green Hills compilers), and Language Processors, Inc. (LPI). Several DOS language compilers, including Lahey and Watcom (with Lattice next), have followed suit by porting their products to the 386 DOS extender environment.

A number of vendors now offer language compilers targeted for 386 DOS extenders. Table 1 shows a listing by language of the vendors that offer compilers. Table 2 provides a vendor list showing company names, addresses, and phone numbers.

WRITING A COMPILER?

Even though I've reviewed 386 compilers over the years, I was surprised to discover that a handful of programmers write most of the language compilers used for development. In the minicomputer and Unix workstation market, you'll find firms like MetaWare, SVS, Green Hills, and LPI mentioned again and again. Watcom (from Canada) provides both mainframe and PC compilers. The GNU C and C++ compiler authored by Richard Stallman and volunteers from the Free Software Foundation are also found on several different processors. In Europe you'll find a similar situation, with only a few firms

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

like Prospero or Salford actively targeting US markets.

In my conversations with the authors of several compilers, I've been intrigued to learn what language the experts use for writing compilers. Maybe you'll be surprised to discover that C is generally not the language of choice. Instead, Pascal is the overwhelming favorite.

Although C has become the market leader for developing applications, the compiler writers beg to differ. I presume that part of the answer lies in what language that the compiler writers learned first, just as the first word processor or program editor I used has warped my judgment ever since. Still, some of these decisions are also based on language features.

The compiler writers from MetaWare, SVS, Green Hills, and Prospero graduated from college at a time when Pascal was the language in vogue. All these firms write all their compilers in Pascal. The dark horse in the Unix crowd is LPI, with compilers written in

TABLE 1: Languages Available for 386 DOS Extender Environments

Language	Vendor	Product
ADA:	Alsys	Alsys Ada 386
	RR Software	386/ADA
	Telesoft	Telesoft-Ada
APL:	STSC, Inc.	APL-PLUS II
	dyadic	dyalog APL/386
BASIC:	Language Processors (LPI)	LPI-BASIC
	Silicon Valley Software (SVS)	SVS 386/BASIC PLUS
	TransEra Corp.	HTBasic
C:	Language Processors (LPI)	NEW C
	MetaWare	High C-386
	MicroWay, Inc.	NDP C-386
	OASYS (Green Hills)	C-386
	Silicon Valley Software (SVS)	SVS 386/C
	Watcom Systems, Inc.	Watcom C 8.0/386
C++:	INTEK	C++
	MicroWay, Inc.	NDP C++
COBOL:	Language Processors (LPI)	LPI Cobol
FORTH:	Laboratory Microsystems (LMI)	UR/FORTH
FORTRAN:	Lahey Computer Systems	F77L-EM/32
	Language Processors (LPI)	LPI-FORTRAN
	MicroWay, Inc.	NDP Fortran-386/486
	OASYS (Green Hills)	Fortran-386
	OTG Systems, Inc.	FTN77/386
	Science Applications (SAIC)	SVS FORTRAN-386
	Silicon Valley Software (SVS)	SVS 386/FORTRAN
	Watcom Systems, Inc.	FORTRAN 77/386
LISP:	Symbolics, Inc.	CLOE-386
PASCAL:	MetaWare	Professionsal Pascal-386
	MicroWay, Inc.	NDP Pascal-386
	OASYS (Green Hills)	Pascal-386
	Science Applications (SAIC)	SVS Pascal-386
	Silicon Valley Software (SVS)	SVS 386/Pascal
PL/I:	Language Processors (LPI)	LPI-PL/I
PROLOG:	Epsilon	MProlog
	Expert Systems Int'l, Inc.	Prolog-2
SMALLTALK:	ParcPlace Systems	Smalltalk-80/386
SPITBOL:	Spitbol-68K/386	Catspaw

Parallel Programming for "C"

INTERWORK™

A Concurrent Programming Toolkit

Interwork is a "C" program library which allows you to write your programs as a set of cooperating concurrent tasks. Very useful for real-time applications, simulation, and parallel programming.

New version 3.0 now available

FEATURES

- Unlimited number of tasks
- Fast context switching
- **NEW!** Optional time sliced scheduling
- Full inter-task communication
- **NEW!** Enhanced interrupt and trap handling
- **NEW!** Expanded user's guide with examples

Interwork is available for the following systems:

IBM PC, XT, AT	PC-DOS 2.0 or later	$189
IBM PC AT	XENIX 286	$239
Intel 386	UNIX V.3/386	$329
DEC VAX™ SUN 3,4	UNIX 4.2BSD	$329

For more information please contact:

Block Island Technologies

Innovative Computer Software

15455 NW Greenbrier Parkway, Suite 210
Beaverton, Oregon 97006
(503) 690-7181 FAX (503) 645-7732

Circle **244** on Reader Service Card

— WRITE MODE ZERO —

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A professional workstation environment for OS/2®

Hamilton C shell™

"...much more powerful than CMD.EXE...blindingly fast... we have a winner...a much-needed and well-done product."
– Personal Workstation Magazine

The superior alternative to the standard OS/2 command processor. Faithfully recreates the entire C shell language as described in the Berkeley 4.3 *UNIX® Programmer's Manual*. Created explicitly for OS/2. Not one line ported from or created on anything but OS/2. Extensively multithreaded.

Features: Command line editing • History • Filename and command completion • Arrow and function keys • Enormous 64KByte command lines • Aliases and shell procedures • PATH hashing • Recursive filename wildcarding • Fully nestable control structures • Powerful expression grammar • Command substitution • Background threads and processes.

Over 113 commands: `alias, cat, chmod, cls, cp, cut, diff, dirs, du, eval, fgrep, grep, hashstat, head, history, label, ls, kill, markexe, more, mv, popd, ps, pushd, pwd, rm, sed, sleep, split, strings, tabs, tail, tar, tee, time, touch, tr, uniq, vol, wait, wc` and others.

Supports HPFS and long filenames.

Requires OS/2 1.1 or later. All executables will run properly in a Presentation Manager window. Not copy-protected.

$350.00. Unconditional satisfaction guarantee. MasterCard & Visa accepted. ($365.00 Canada/Mexico; $395.00 elsewhere.)

Hamilton Laboratories
13 Old Farm Road, Wayland, MA 01778–3117, U.S.A.
Phone 508-358-5715 • FAX 508-358-1113 • BIX hamilton

Circle **334** on Reader Service Card

PL/I. That dates their principal programmers a bit.

If you talk with Tom Pennello (MetaWare) or Steven Glanville (SVS), they will gladly enumerate the advantages of Pascal over C. K&R C was not well thought out in the early days—no enums, poor support for structure passing, and no prototypes. Pascal provides much better support for catching errors at compilation. For example, Pascal supports type checking and can catch the wrong order and number of parameters to procedure calls.

Note that all these firms author their own Pascal compilers, which include enhancements far above the ISO Pascal standard—at least with an *address* and *pointer* operator. First, you port your Pascal compiler and runtime to the new RISC processor or microprocessor. Then you just follow with the other languages you support.

If only life were so easy for us mere mortal programmers. PJ

Experts Always Agree on the Solution!

However, they rarely agree on the features that make Salford's FTN77/386 superior!

"The defaults for the compiler are easily configured, and the compiler supports a load and go option for quick testing".
. . . Steve Baker - "Programmer's Journal ", July 1990

"The debugger was very easy to use. There is an on-line help facility which made reference to the manual unnecessary".
. . . Mike Gunn and Arul Britto, ".EXE", May 1989

"FTN77/386 provides more intrinsic functions than any other: functions to manipulate string, handle screen and keyboard I/O, read the date and time, parse the command line . . . Advanced file manipulation functions allows an FTN77/386 program to work with files in a C-like manner".
. . . Scott Ladd, "Computer Language", Nov. 1988

"For sheer speed and sleekness, the OTG compiler stands out like a Ferrari among Yugos".
. . . Jack W. Crenshaw, "Computer Language", May 1990

•Important Benchmark Results•

73% - of our new business in 1989 came through existing customer references
99.6% - satisfaction rating from our existing accounts
100% - guaranteed satisfaction

OTG SYSTEMS, INC.

Clifford, PA • (717) 222-9100 • Fax: (717) 222-9103

SERVICE, QUALITY AND INTEGRITY
FTN77/386 Developed by Salford Software Marketing

Circle **270** on Reader Service Card

SUBSCRIBE NOW

A Difference That Must be Seen!

Diff_it™

The difference program that you've been waiting for, you've seen the others, now see the difference.

- *Unlimited file sizes.*
- *Scrollable split screen display.*
- *Display in ascii or hex.*
- *Configurable compare algorithm.*
- *25, 43 and 50 line display modes.*
- *Command line interface.*
- *Pop-up menu interface.*
- *Pop-up help screens.*
- *Optional output to disk file.*
- *Fast Terse mode.*
- *Line, Word and Byte compare.*
- *Batch File drive-able.*

Requires IBM PC or 100% compatable, 148KB memory, DOS 2.0 or higher, floppy or hard disk. Not copy protected. 30 day money back guarantee, send check or money order. $129.95

ORDER NOW!

DOUBLE R SOFTWARE, INC.
RD 1, Fifth Ave, Valley Forge Manor
Phoenixville, PA 19460
(215) 666-6104

Contact this advertiser directly.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TABLE 2: Company References

Alsys
1432 Main Street
Waltham, MA 02154
(617) 890-0030

Catspaw
P.O. Box 1123
Salida, CO 81201
(719) 539-3884

Cyrix Corporation
1761 International Parkway, # 135
Richardson, Texas 75081
(214) 234-8387

Dyadic
Park House, The High Street
Alton, Hampshire
England
FAX: (44)420-89886

Ergo Computing
One Intercontinental Way
Peabody, MA 01960
(508) 535-7512

Epsilon
Kurfurstendamm 188, D-1000
Berlin 15, W. Germany
FAX: (49)308823594

Expert Systems
7 W. Way, Unit 14
Botley, Oxford OX2-0JB
England
FAX: (44)865250270

INTEK
1400 112th Avenue, SE
Bellevue, WA 98004
(206) 455-9935

Laboratory Microsystems, Inc (LMI)
3007 Washington Blvd
Marina del Rey, CA 90295
(213) 306-7412

Language Processors Inc (LPI)
959 Concord Street
Framingham, MA 01701
(508) 626-0006

Lahey Computer Systems, Inc.
P.O. Box 6091
Incline Village, NV 89450
(800) 548-4778

MetaWare
2161 Delaware Avenue
Santa Cruz, CA 95060
(408) 429-6382

MicroWay, Inc.
Cordage Park, Bldg 20
Plymouth, MA 02360
(508) 746-7341

OASYS
230 Second Avenue
Waltham, MA 02154
(617) 890-7889

OTG Systems, Inc.
P.O. Box 239
Clifford, PA 18413-0239
(717) 222-9100

ParcPlace
2400 Geng Road
Palo Alto, CA 94303
(415) 859-1000

Phar Lap Software
60 Aberdeen Avenue
Cambridge, MA 02138
(617) 661-1510

RR Software
2317 International Lane
Madison, WI 53704
(608) 244-6436

Silicon Valley Software (SVS)
1710 S. Amphlett Blvd, Suite 100
San Mateo, CA 94402
(415) 572-8800

STSC, Inc.
2115 E. Jefferson Street
Rockville, MD 20852
(800) 592-0050

Symbolics
#8 New England Executive Park
Burlington, MA 01803
(617) 621-7500

Talesoft
5959 Cornerstone Court West
San Diego, CA 92121
(619) 457-2700

TransEra
3707 N. Canyon Road
Provo, UT 84604
(801) 224-6550

Watcom
415 Phillips Street
Waterloo, Ontario
Canada N2L 3X2
(519) 884-2700

GUIDO

C Library for Graphical User Interface

GUIDO is a powerful set of C user interface functions for use in a graphical environment. GUIDO's high level objects make it easy to build intuitive front ends for your software.

- Pull-down, pop-up, vertical and horizontal menus
- Movable, overlapping, windows, data entry, text and list boxes
- Radio buttons, percent bars, prompt boxes and more
- Turbo C, Microsoft C and Quick C
- EGA and VGA support
- Object oriented windowing environment
- User definable High Level Objects

Price $249 with source $499

Other Products

Essential Graphics
Essential Communications
C-Utility Library
Essential B-Tree
/*resident_C*/
BreakOut II
Hold Everything

South Mountain Software
76 S. Orange Ave., Suite 3
South Orange, N.J. 07079
NJ 201-762-6965
Fax 201-762-0118

Essential Communications, Essential Graphics, C-Utility Library, Essential B-Tree, /*resident_C*/, Hold Everything, BreakOut II and GUIDO are trademarks of South Mountain Software, Inc.

SOUTH MOUNTAIN SOFTWARE™

1-800-451-6174

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

By John Otken

Windowing text with X

Using the X Window protocol for portability

I have written a number of windowing systems for assembly language programs, with varying degrees of satisfaction. The first of these was a static windowing system for a CP/M debugger. Although it was simple, it just wasn't flexible.

The next system was more powerful, with overlapping windows and scrolling. Initially, however, my algorithms for overlapping windows used too much memory, and scrolling overlapped windows was a programming nightmare. I found methods for efficient window overlapping and scrolling in an article called "Graphics in Overlapping Bitmap Layers" by Bell Laboratories' Rob Pike in the July 1983 *ACM Transactions On Graphics*.

I ran across a C implementation of Pike's algorithms by Simon Kenyon. All I had to do then was translate and adapt the code to 80x86 assembly language. The code worked well, with almost no performance penalty for text and scrolling operations on overlapped windows. I eventually abandoned the code during a switch to a large model dynamic storage system since the code was difficult to maintain on this system. The algorithms needed reworking, and I didn't have the time to spare. The next window system was easier to maintain because it lacked support for scrolling and overlapping windows. Easy as it was, I never liked it.

John Otken is president of Soft Advances where he hacks assembly language into debuggers and other software for embedded systems. You can leave him mail on his BBS: (213) 559-1449.

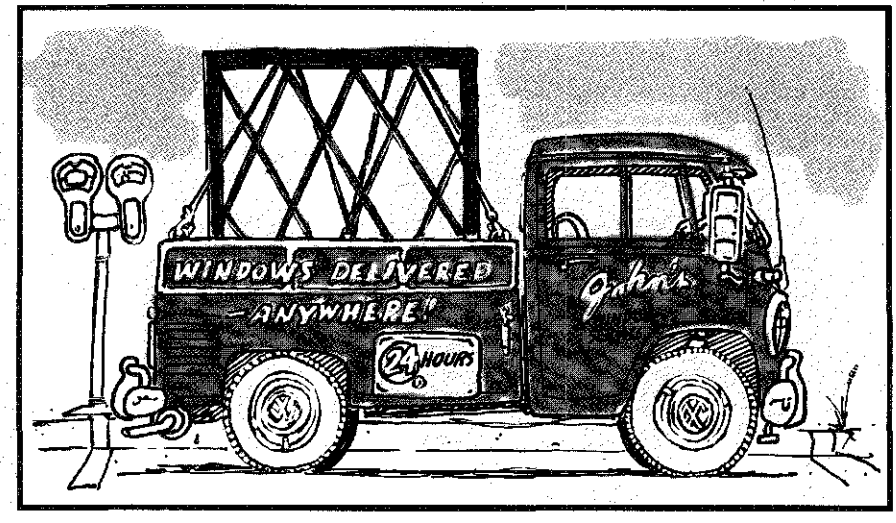
THE X WINDOW SYSTEM

Later I learned about the X Window System (or simply X) developed at MIT and popular in the Unix community. X was designed as part of a debugging environment in multi-process distributed applications and as device-independent support for bitmap displays.

The result is a window system where the window server and client application execute as separate tasks and communicate over a full duplex network session. The server and client tasks do not have to run on the same computer or even on the same type of CPU. A client application on a Cray supercomputer might communicate with a server on a Sun workstation. Because an X server supports connections to multiple clients, a single workstation can display windows from applications running on several different computers. Remote displays can be more efficient than network file systems as, for example, in a large data base application that does much more file I/O than screen I/O.

An interesting notion in X is support for window hierarchies. Every window has a parent window. A child window must fit within the parent window and its position is relative to the parent's origin. If the parent moves, so do all its progeny. Children of the root window are known as "top-level." Hierarchical windows simplify a user interface because scroll bars and other window control structures become subwindows. When a parent window moves, so do the scroll bars. When the mouse moves into the scroll bar window, the server can send a generic mouse change message, making special code to detect when the mouse enters a scroll bar unnecessary.

X is a "primitive" windowing system, without preset functions for scroll bars and dialog boxes. A window manager such as OSF/Motif implements these facilities using primitive X functions. A client application asks Motif to open a window with scroll bars, and Motif sends multiple X functions to open a main window and subwindows for the scroll bars. Before I get too far with this introduction, I should warn you that I am no X-pert, and some of my



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Listing 1

```
; (c) 1990 Soft Advances, all rights reserved.
                title   W window include file

NULL_PARENT     equu    0
NULL_SIBLING    equu    0
NULL_WINDOW     equu    0
WINDOW_MAX      equu    50
TEXT_MAX        equu    128


; color equates
;
BLACK_PIXEL     equu    0
WHITE_PIXEL     equu    1


; request codes
;
W_CreateWindow  equu    1
W_DestroyWindow equu    4
W_Text          equu    74

; reply codes
;
W_Error         equu    0
W_Reply         equu    not W_Error

; reply error codes
;
BadRequest      equu    1               ; bad request code
BadValue        equu    2               ; integer parameter out of range
BadWindow       equu    3               ; parameter not a window
BadAlloc        equu    11              ; insufficient resources
BadColor        equu    12              ; no such color


; reply structure definitions
;
_WRep struc                             ; server reply/event
  WRep_code     dw      ?
  WRep_p1       dw      ?
  WRep_p2       dw      ?
  WRep_p3       dw      ?
  WRep_p4       dw      ?
  WRep_p5       dw      ?
_WRep ends

SIZEOF_WREP     equu    (size _WRep)

; error reply
;
WRep_error_code equu    WRep_p1

; CreateWindow
;
WRep_windowId   equu    WRep_p1


; request structure definitions
;
_WReq struc
  WReq_code     db      ?
  WReq_func     db      ?
  WReq_length   dw      ?
  WReq_WindowId dw      ?
  WReq_p1       dw      ?
  WReq_p2       dw      ?
  WReq_p3       dw      ?
  WReq_p4       dw      ?
  WReq_text     db      TEXT_MAX dup(?)
_WReq ends

SIZEOF_WREQ     equu    (size _WReq - size WReq_text)

; CreateWindow
;
WReq_y_x                equu    WReq_p1
WReq_height_width       equu    WReq_p2

; Text
;
WReq_count      equu    WReq_p2
WReq_attribute  equu    WReq_p3
```

Listing 2

```
        title   wlib - assembly language window interface
        include window.inc
        include asm.inc

        .data?
WReq    _WReq   < >                     ; window request packet
WRep    _WRep   < >                     ; reply/event packet


        .code
        extn    transmit_window_request,receive_window_reply,video_preset


;;      get reply
;
;       exit    DS:SI   reply packet from server
;               Cf      if error
;       uses    AX
;
get_reply proc
        pushm   cx,ss
        pop     ds
        lea     si,WRep[bp]
        mov     cx,SIZEOF_WREP
        call    receive_window_reply
        pop     cx
        ret
get_reply endp


;;      request packet read
;
;       exit    DS:SI   request packet
;
request_packet_read proc
        push    ss
        pop     ds
        lea     si,WReq[bp]
        ret
request_packet_read endp


;;      request packet write
;
;       exit    ES:DI   request packet
;
request_packet_write proc
        push    ss
        pop     es
        lea     di,WReq[bp]
        ret
request_packet_write endp


;;      send long request
;
;       entry   CX      text byte count
;               DS:SI   request packet
;       uses    AX
;
send_long_request proc
        push    cx
        add     cx,SIZEOF_WREQ
        call    transmit_window_request
        pop     cx
        ret
send_long_request endp


;;      send short request
;
;       uses    AX
;
send_short_request proc
        push    cx
        mov     cx,SIZEOF_WREQ
        call    transmit_window_request
        pop     cx
        ret
send_short_request endp


;;      WCreateWindow
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;
;       entry   BX      parent window
;               CL      x
;               CH      y
;               DL      width  (> 0)
;               DH      height (> 0)
;       exit    BX      window handle
;               Cf      if error (no memory, too many windows)
;       uses    AX
;       note    border width is 1
;
WCreateWindow proc
        pushm   si,ds
        call    request_packet_read
        mov     WReq_code[si],W_CreateWindow
        mov     WReq_WindowId[si],bx    ;  (set parent window)
        mov     WReq_y_x[si],cx
        mov     WReq_height_width[si],dx
        call    send_short_request

        call    get_reply
        jc      wcw1                    ; if request failed
        mov     bx,WRep_windowId[si]
wcw1:   popm    ds,si
        ret
WCreateWindow endp


;;      WDestroyWindow
;
;       entry   BX      window handle
;               Cf      if bad window
;       uses    AX
;
WDestroyWindow proc
        pushm   si,ds
        call    request_packet_read
        mov     WReq_code[si],W_DestroyWindow
        mov     WReq_WindowId[si],bx
        call    send_short_request
        popm    ds,si
        ret
WDestroyWindow endp


;;      WPreset
;
;       may use AX,BX,CX,DX,DI,SI,DS,ES
;
WPreset proc
        call    video_preset
        ret
WPreset endp


;;      WDrawText
;
;       entry   AL      attribute
;               BX      windowId
;               CX      string byte count
;               DL      x coordinate
;               DH      y coordinate
;               DS:SI   string pointer
;       uses    AX
;
WDrawText   proc
        pushm   di,si,ds,es
        call    request_packet_write    ; copy string to request packet
        lea     di,WReq_text[di]
        push    cx
        rep     movsb
        pop     cx

        call    request_packet_read     ; build WDrawText packet
        mov     WReq_code[si],W_Text
        mov     WReq_WindowId[si],bx
        mov     WReq_count[si],cx
        mov     WReq_y_x[si],dx
        mov     ah,0
        mov     WReq_attribute[si],ax
        call    send_long_request
        popm    es,ds,si,di
        ret
WDrawText endp
        end
```

Listing 3

```
        title   window server
        include window.inc
        include asm.inc

; server window structure
;
window_str struc
  w_origin_x    db      ?
  w_origin_y    db      ?
  w_corner_x    db      ?
  w_corner_y    db      ?
  w_parent      dw      ?
  w_sibling     dw      ?
  w_child       dw      ?
window_str ends
;
w_origin        equ     word ptr w_origin_x
w_corner        equ     word ptr w_corner_x


; local variables for WDrawText and clipping procedures
;
ct_locals struc
  ct_offset     dw      ?               ; text pointer offset
  ct_segment    dw      ?               ;              segment
  ct_windowId   dw      ?               ; active window handle
  ct_attribute  dw      ?
ct_locals ends


        .data?

w_pointers      dd      WINDOW_MAX dup(?)
w_count         dw      ?               ; number of open windows

root_child      dw      ?               ; first child for root window


WRep_s          _WRep   < >
WReq_s          _WReq   < >


        .code
 extn   save_most,calloc_read,receive_window_request,transmit_window_reply
 extn   write_video_memory


;;      alloc window struct
;
;       exit    AX      windowId
;               DS:SI   window structure
;               Cf      if no memory or too many windows
;
alloc_window_struct proc
        mov     ax,w_count[bp]          ; check window count
        cmp     ax,WINDOW_MAX
        jae     aws2                    ;  if too many windows

        inc     ax                      ; update window count
        pushm   ax,cx,di
        mov     di,ax
        mov     w_count[bp],ax

        mov     cx,size window_str      ; allocate window structure
        call    calloc_read
        jc      aws1                    ;  if no memory

        add     di,di                   ; put structure address in table
        add     di,di
        mov     wptr w_pointers[bp+di],si
        mov     wptr w_pointers[bp+di+2],ds
aws1:   popm    di,cx,ax
        ret

aws2:   stc
        ret
alloc_window_struct endp


;;      clip aunt text
;
;       entry   BX      windowId (most distant grandparent)
;               CX      byte count
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

interpretations may be in error. Still, I have done a lot of windows (programming, that is).

THE X SYSTEM MODEL

X is designed as a network display system with two components: a display and an application. The display is called a server; the application, a client. A server directly controls each physical display and its associated keyboard and mouse. A client indirectly controls one or more windows on the X server by sending "requests" to the server. The server sends "replies" to client requests and notifies the client of "events" such as mouse movement and keyboard activity.

The client and server communicate over a reliable duplex byte stream using a simple block stream protocol. The byte stream may use a network connection or an inter-process communication facility. For efficient operation, messages between client and server are asynchronous. When a client wants to do window operations, it sends one or more requests to the server. Typical requests include commands to display text and to open or close a window.

The server sends reply and event messages to the client as necessary. A reply message might notify the client of an error, while an event message may result from keyboard or mouse activity.

THE X PROTOCOL

The X protocol defines the format of messages on the byte stream between a client and server. Messages in both directions are variable length, depending on the function and the amount of text or graphics data to be transferred. The C structure definition for a client request to create a window is:

```
struct {
      unsigned char reqType;
      unsigned char data;      /* depth */
      unsigned short length;   /* in 4-bytes */
      Window windowId, parent;
      short x, y;    /* relative coordinates */
      unsigned short width, height;
      ... other parameters
} xCreateWindowReq;
```

To open a window, a client builds a request with *reqType* set to 1, *windowId* set to the value passed, *parent* set to the parent window handle, and additional parameters set to window coordinates, height, width, depth. When the server receives the request, it opens the window and sends a "reply" (a CreateNotify event) to the client. The C structure for a reply is:

```
struct {
      unsigned char type;
      unsigned char data1;
      unsigned short SequenceNumber;
      unsigned long length; /* in 4-bytes */
      unsigned long data[6];
} xGenericReply;
```

The server sets *type* to non-zero for a normal reply and 0 for an error. If the request fails, the server returns an error code. Reply codes other than 0 are known as events. Errors and events use the same structure definition. The keyboard, mouse buttons, and mouse movement are the most common sources of events. Events return values for the time, event window, and mouse coordinates in the parameters. When all or part of an obscured window becomes visible (window "exposure"), the server sends the client an exposure event. When a client receives such an event, it must redraw the window. This design allows the server to conserve memory because it does not have to save data hidden under windows. Of course, a well-endowed server can always save window contents and forego the exposure event.

THE X LIBRARY

X clients do not build request packets directly, since this would entail lots of work and hinder their portability. X provides a C language interface called XLIB that isolates the protocol details from the program and provides a reasonable interface.

The XLIB function to create a window is **XCreateWindow(****display, parent, x, y, width, height, borderwidth, depth,* ...). This function returns the new window handle or 0 if the function fails. X does not display the window until it has been "mapped" with another function, **XMapWindow**. The function used to "destroy" a window is **XDestroyWindow(****display, window*). When an X window dies, its descendants die, too. A window manager thus does not have to close subwindows such as scroll bars.

The XLIB function used to display text is **XDrawText(****display, window, x, y, text_items, nitems*). The *display* parameter identifies the server to communicate with.

Just as many assembly language library routines are adapted from the

New Version 3.0

Commenting Disassembler!

SOURCER™ 486

■ **SEE HOW PROGRAMS WORK**
■ **EASILY MODIFY PROGRAMS**

SOURCER™ creates detailed commented source code and listings from memory and executable files directly suitable for reassembly. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088 through 80486, 8087 to 80387, and V20/V30 instruction sets.

We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER. "Sourcer is the best disassembler we've ever seen." PC Magazine 1/17/89 page 101.

BIOS SOURCE

for PS/2, AT, XT, PC and Clones

■ **CHANGE AND ADD FEATURES**
■ **CLARIFY INTERFACES**

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

ASM ProPak™

■ **ASSEMBLY TOOLS PACKAGE**

Save $50 with the ASM ProPak™ assembly tools package. It includes Sourcer with the BIOS Pre-Processor; Unpacker,™ for unpacking packed EXE files; ASMtool,™ to automatically generate flowcharts and tree diagrams from assembly source code; and View-It,™ to quickly view files of any size and see up to 8 times more information on a single screen using EGA/VGA equipment.

Sourcer–Disassembler	$129.95
BIOS Pre-Processor	49.95
Unpacker–Unpacks packed files	39.95
ASMtool–Source code analyzer	89.95
View-It–Zoomable file viewer	69.95
ASM ProPak–All Above (save $50)	329.75
Sourcer w/BIOS–(save $10)	169.95

Shipping & Handling: USA $5; Canada/Mexico $10; Other Countries $15; CA Residents add sales tax. Visa/MasterCard/COD accepted.

NOT COPY PROTECTED
30-DAY MONEY-BACK GUARANTEE

For orders or information call:

1-800-662-8266

V Communications Incorporated
4320 Stevens Creek Blvd., Suite 275-PJ
San Jose, CA 95129
(408) 296-4224

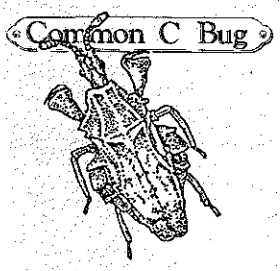
Circle 125 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

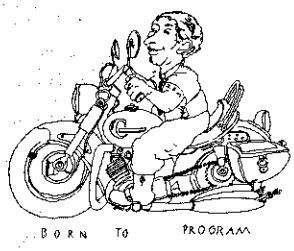
Multi-color imprints on the highest quality 100% cotton T-shirts.

SIZES: S, M, L, XL, XXL

$14.95 each — includes U.S. shipping



COMMON C BUG



BORN TO PROGRAM

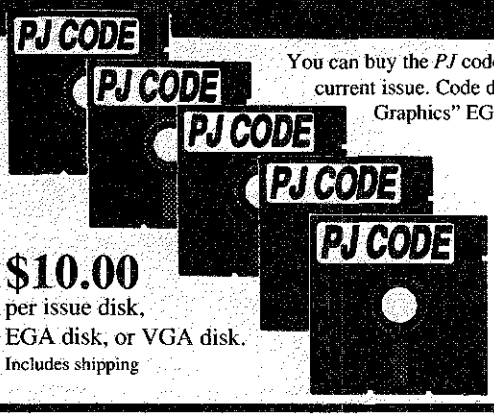
PARALLEL PORTS

RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

All mail orders must be accompanied by check or Money Order – PREPAID ONLY.

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



You can buy the *PJ* code listings from volume 5.5 to the current issue. Code disks for the Michael Abrash "On Graphics" EGA/VGA series are also available.

$10.00 per issue disk, EGA disk, or VGA disk. Includes shipping

Diskettes are available for the following isssues:

EGA	6.1	7.1	8.1
	6.2	7.2	8.2
VGA	6.3	7.3	8.3
	6.4	7.4	8.4
5.5	6.5	7.5	8.5
5.6	6.6	7.6	

360K MS-DOS disks standard; other formats are available.

PJ ISSUE SPECIAL!

$4.00*ea.
Regularly $7.00

SAVE 43% ON OVERSTOCKED BACK ISSUES

Do you need issues 2.4, 3.2, 4.6, 5.4, 6.2, 6.6, 7.2, 7.4, 8.1, or 8.2 to complete your library? They'll never be cheaper — so now's the time to get 'em!

*Plus $1.50 handling charge per order and $2.50 shipping per magazine to Canada or Mexico, $5.00 shipping per magazine to any other foreign destination. Offer expires 10-31-90.

Vol. 4.6: DTACK GROUNDED; Unix Books, Part 1; How many amoebas at one o'clock; DOS functions 45H and 46H; DOS Terminate ans Stay Resident; Turbo Pascal inline code; Pascal—Assembler souce code for DOS function calls; Alignment problems in C and Managing memory allocation; Proposed ANSI C Language Standard, Part 4; The Canon A1 laser printer; Genifer dBASE applications generator reviewed; Advanced Trace86 debugger reviewed.

Vol. 5.4: Starting your own software house—technical support; Unix books, Part 3; Consultant productivity and effectiveness; EGA set/reset capabilities; ANSI C language standard; Character tables and string functions in C; MWWIND—a menu with windows utility; Variable passing and hiding in QuickBASIC 2 subprograms.

Vol. 6.2: Hal Hardenbergh on RISCs, CISCs, and Fabs; Pink noise; Yet another VGA write mode; Inside Turbo Pascal 4.0; The proposed ANSI C Language Standard; Converting a Turbo C program to a TSR; 80286 and 80386 programming; 80386 software development; The Norton Guide for BIOS/DOS Assembly.

Vol. 6.6: PC DOS 4.0; OS/2 vs. UNIX; VGA screen saving and color selection; MASM quirks; addressing extra memory for DOS; DOS device drivers debugging and information utility; proposed ANSI C language standard; Ray Duncan on Forth; Ada data types and exceptions; PC floating-point hardware and software.

Vol. 7.2: OPERATING SYSTEMS: Undocumented DOS features; fast 80386 block moves under DOS; OS/2 dynamic link libraries; OS/2 multitasking; debugging utility for PM and Windows; EGA and VGA animation; disk formatting in Pascal; reviews of windowing and graphics tools, disassemblers, and *The 80386 Book*.

Vol. 7.4: COMMUNICATIONS: A simple terminal program in PM; communications and protocols; database normalization theory and practice; Turbo Pascal 5.5 development tools; Lint utilities for the PC; a millisecond timer for BASIC; reviews of RamNet, DOS/16M, PCX *Technical Reference Manual*, and Programmer's Toolkit.

Vol. 8.1: NETWORKING: Network File System; an overview of NetBIOS on the PC, from *C Programmer's Guide to NetBIOS* by W. David Schwaderer; fast circle drawing on the VGA; Dynamic Data Exchange in Windows; DESQview API features; computing time in C; a menu system with Turbo Pascal 5.5.

Vol. 8.2: GRAPHICS: Faster circles for the VGA; the TIGA interface from TI; the EGA palette; 800x600x16 Super-VGA mode programming; Dynamic Data Exchange for communications; manipulating large data; fast file I/O in QuickBASIC; industry news; 1989 article index.

A few of the Back Issues currently available

$7.00 each PREPAID ONLY

Vol. 8.3: OPERATING ENVIRONMENTS: Undocumented DOS shell; graphics programming in Windows; mousing around; messages in PM; accessing high memory; ASM:advanced segmentation; ellipses on the VGA; up OOP alley in Pascal.

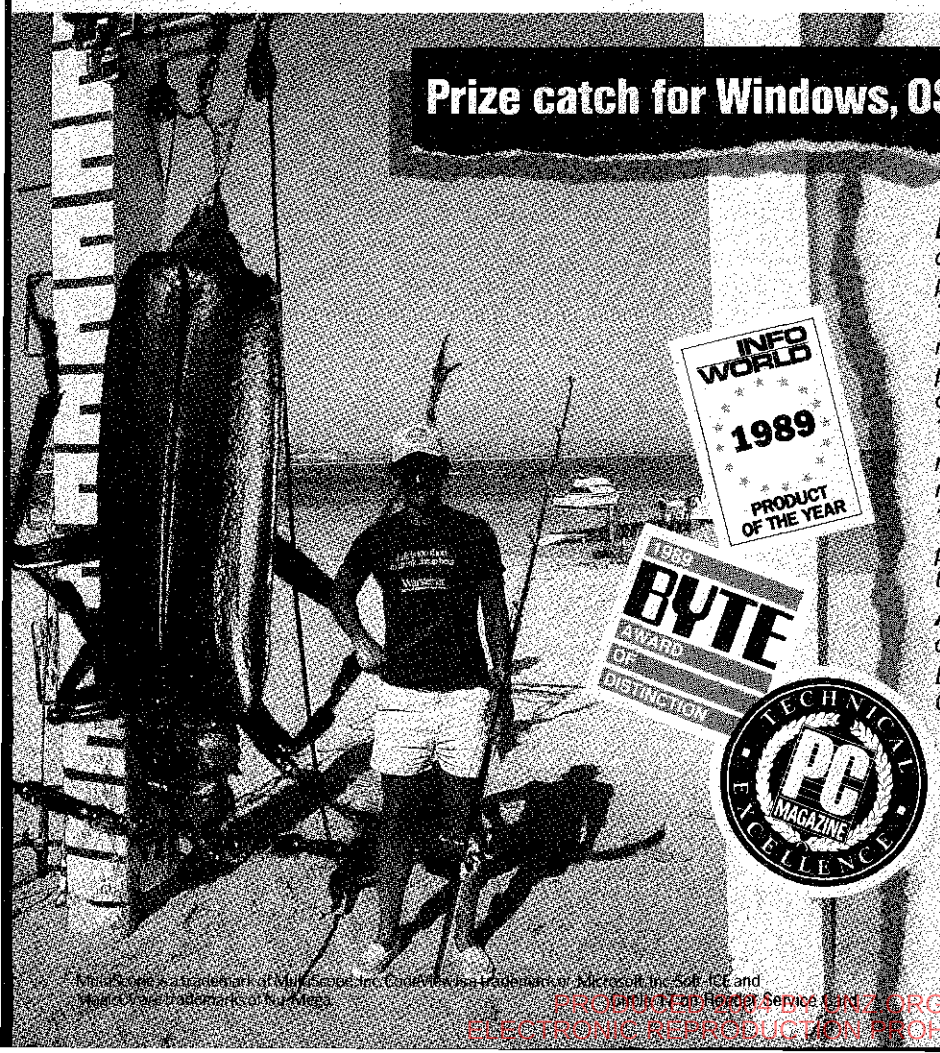
Vol. 8.4: PROGRAMMING IN THE PROTECTED MODE: Faster VGA ellipses and more; running unprotected virtual-8086 mode; converting to 386 protected mode; OS/2 2.0 memory models for the 386; Fortran for the 386; using C++ features in C; compensating for system clock idiosyncracies; ASM: errors and stream I/O; Cobol-85 nested; deliberately breaking the rules.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
;               DL      column
;               DH      row
;               DS:SI   window structure for BX
;               BP+DI   local variables
;       uses    AX,BX,CX,DX,SI,DS,DI
;
clip_aunt_text proc
cat1:   mov     bx,w_sibling[si]        ; process overlapping siblings of
        cmp     bx,NULL_SIBLING         ;  most distant grandparent
        je      cat4                    ;  (if no more siblings)
        call    read_window_struct

        cmp     dh,w_origin_y[si]       ; check for overlapping window
        jb      cat1                    ;  if no overlap
        cmp     dh,w_corner_y[si]
        jae     cat1                    ;  if no overlap
        cmp     dl,w_corner_x[si]
        jae     cat1                    ;  if no overlap

        mov     ax,cx
        add     al,dl
        cmp     al,w_origin_x[si]
        jb      cat1                    ;  if no overlap
        cmp     dl,w_origin_x[si]
        jae     cat3                    ;  if left overlap
        cmp     al,w_corner_x[si]
        jbe     cat2                    ;  if right overlap

        pushm   ax,bx,cx,dx,di,si,ds    ;  else middle overlap
        push    ct_offset[bp+di]        ;   save current state and process
        call    cat3                    ;   exposed right side line
        popm    ct_offset[bp+di]        ;   fragment, then restore and
        popm    ds,si,di,dx,cx,bx,ax    ;   process exposed left side line
                                        ;   fragment

cat2:   sub     al,w_origin_x[si]       ; here for right overlap
        sub     cx,ax
        jmp     cat1

cat3:   mov     al,w_corner_x[si]       ; here for left overlap
        sub     al,dl
        jbe     cat5                    ;  if complete overlap
        add     dl,al                   ;  set new X coordinate DL=corner_x
        sub     cx,ax                   ;  adjust byte count and text offset
        add     ct_offset[bp+di],ax
        jmp     cat1

cat4:   mov     ax,ct_attribute[bp+di]  ; write text to display memory
        lds     si,dptr ct_offset[bp+di]
        jmp     write_video_memory

cat5:   ret
clip_aunt_text endp


;;      clip parent text
;
;       entry   CX      byte count
;               DL      column
;               DH      row
;               BP+DI   local variables
;       uses    AX,BX,CX,DX,SI,DS,DI
;
clip_parent_text proc
        mov     bx,ct_windowId[bp+di]
        call    read_window_struct

cpt1:   mov     ax,w_parent[si]
        cmp     ax,NULL_PARENT
```

continued



Prize catch for Windows, OS/2 and DOS developers.

Bugs take note: MultiScope™ is the only debugger that is CodeView™ compatible and provides up to 13 views of programs.

The only debugger to offer run-time and post-mortem debugging. The only debugger to provide Windows, Presentation Manager and character mode user interfaces.

The only debugger to incorporate 386 virtual machine and ICE technologies from Nu-Mega, makers of Soft-ICE™ and MagicCV.™

The only debugger "that lets you focus on the problem to be solved rather than the tool used to solve it." –IEEE Software 11/89

And the only debugger to win top industry awards from PC Magazine, Info World and BYTE.

*DOS–**$179**; DOS & Windows–**$379**; OS/2–**$449**.*

*For your free trial-version, please call **800-999-8846**, (415) 323-4286, or FAX (415) 323-4186. For European pricing, please call ++46-13-63189.*

INFO WORLD 1989 PRODUCT OF THE YEAR

BYTE 1989 AWARD OF DISTINCTION

TECHNICAL EXCELLENCE PC MAGAZINE

MULTISCOPE™ DEBUGGER

The World's Best Debuggers

MultiScope, Inc., a Logitech Company

MultiScope is a trademark of MultiScope Inc. CodeView is a trademark of Microsoft Inc. Soft-ICE and MagicCV are trademarks of Nu-Mega.

PSCDN@HOTMAIL.COM
ELECTRONIC REPRODUCTION PROHIBITED

CODE RUNNER

Create ultra compact and fast TSRs with full DOS access!

The most powerful tool available for creating feature packed, tiny TSRs in C or assembler. Works with C compilers from Microsoft, Borland, Watcom and Zortec.

CodeRunneR Features

- Auto recovery of Initialization Code and Data
- All functions hand crafted highly optimized assembly language routines
- Function Level Granular
- Create MultiTasking programs
- Scheduler supports > 10,000 events
- Assess TSRs via up to 256, dynamic hot-keys or interrupts
- BCD Floating Point Library
- Source for TSRs: Help, Programmers Calculator, Business Calculator, ASCII/Color, Date/Alarm & Trap/Debug

CodeRunneR $149.00

Call our BBS for FREE sample TSRs, benchmarks & support. See for yourself how powerful and tiny CodeRunneR TSRs can be.
BBS: (508) 875-8009

NEW

PDK#3

- Spawn (EXEC) large programs from a hosting TSR (<1.5 KBytes)
- Page foreground application or hosting TSR to LIM* or Disk
- Pop-up TSRs over graphics screens!
- Save & Restore COM ports, Mouse and Math Co-Processor states.
- Turn ANY existing program into a TSR (without source code -- even Lotus 1-2-3)! * Needs PDK#1

PDK#1

- COM: Ultra fast interrupt driven multichannel COM library with background file transfers
- LIM: Transparently move TSR code &/or data to LIM
- Flexible printer SPOOLer
- Program registration and protection functions

PDK#1&3 $99.00 each

NO ROYALTIES. MONEY BACK offer and source code available.
Order today by Phone, FAX or BBS.
MasterCard, VISA & AMEX accepted.

Microsystems Software, Inc.™
600 Worcester Rd. Framingham, MA 01701
(508) 626-8511 FAX: (508) 626-8515

CodeRunneR is a Registered Trademark of Microsystems Software, 1-2-3 is a Trademark of Lotus.

Circle 40 on Reader Service Card

A windowing system should be adapted from one that already exists.

standard C library, a windowing system should be adapted from one that already exists. Because I am familiar with X, I chose to base a new windowing system on it. Presentation Manager and Microsoft Windows also make good models for windowing systems, though they lack X's network support.

To make this project practical, major simplifications are necessary. I am implementing only those parts of X essential to my application, such as support for overlapping hierarchical windows and a mouse. The big omission is graphic support, since my application is text-based. I don't need network support, but I want it because it's so much fun. When the windowing system is complete, I will write a simplified Motif that supports scroll bars, dialog boxes, and menus.

The source listings with this column contain a very primitive windowing system, called W for lack of a better name. W contains only two functions: **WCreateWindow** opens a new window, and **WDrawText** writes text to a window. It supports overlapping hierarchical windows. With W, each "pixel" is a character. To conserve register space, the maximum window dimensions are 255 by 255 characters. Space (and time) do not permit W to do more, so it lacks routines to set window priority. There also are no mouse or keyboard events, and network support is simulated.

If you need a complete assembly language windowing system, I will discuss two commercial packages in a future article: Spontaneous Assembly from Base Two Development and Power Lib from Quantasm. These packages are not based on X, Presentation Manager, or Windows, but if your time is valuable, it is cheaper to buy than write.

W SOURCE FILES

W consists of four layers: hardware, window control, network, and application interface. The source for each layer is a separate source file. W simplifies the X protocol in several ways. W uses a maximum field width of sixteen bits instead of the thirty-two bits in X. W uses little-endian byte ordering, the native mode of the 80x86. W also doesn't pad data fields such as text strings to thirty-two bits.

Global equates and structure definitions for the window system are in the include file, WINDOW.INC. The reply and request structures are similar to those in X, but structure members are sixteen bits instead of thirty-two. The smaller data objects are much easier to manipulate in 80x86 assembly language. WLIB.ASM defines the application interface and implements the client side of the W protocol. When a program opens a window or displays text, it calls a procedure in this module. The procedure builds a packet according to the W protocol and passes it to the network. The text display function returns immediately after passing the packet to the network. The open window function waits for a reply from the server and returns the new window handle. The primary action of this layer is to move register parameters to and from reply and request structures. WLIB isolates the protocol from the application and presents an easy program interface.

WCreateWindow opens a new window. It accepts parameters for parent handle, relative coordinates, height, and width. It returns a handle for the new window. **WDrawText** writes text to a window. It accepts parameters for attribute, window handle, byte count, target coordinates, and a pointer to the text string. Notice the client is responsible for most of the window "state" information such as cursor position and attribute. This is characteristic of X.

WNET.ASM is the network simulator. It sits between WLIB and WSERVER. In a network environment, WNET would split into two modules: one for the server and one for the client. X clients and servers usually communicate over a TCP/IP connection. NetBIOS could also be used.

WSERVER.ASM holds the core windowing logic. It receives request packets from the network and interprets

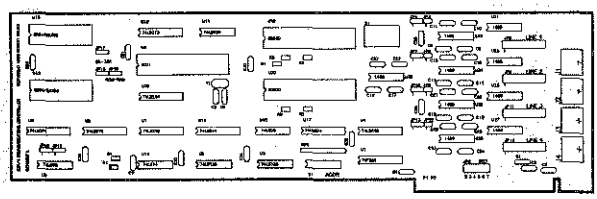
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        je      clip_aunt_text          ;  if no more ancestors
        mov     bx,ax
        call    read_window_struct

        add     dl,w_origin_x[si]
        jc      cpt2                    ;  if outside window limits
        add     dh,w_origin_y[si]
        jc      cpt2                    ;  if outside window limits

        cmp     dh,w_corner_y[si]
        jae     cpt2                    ;  if below parent window
        mov     al,w_corner_x[si]
        cmp     dl,al
        jae     cpt2                    ;  if right of parent window

        sub     al,dl
        mov     ah,0
        cmp     ax,cx
        jae     cpt1                    ;  if line fits in parent window
        mov     cx,ax                   ;  else clip end of line
        jmp     cpt1

cpt2:   ret
clip_parent_text endp


;;      clip sibling text
;
;       entry   CX      byte count
;               DL      column
;               DH      row
;               DS:SI   active window structure
;               BP+DI   local variables
;       uses    AX,BX,CX,DX,SI,DS,DI
;
clip_sibling_text proc
cbt1:   mov     bx,w_sibling[si]
        cmp     bx,NULL_SIBLING
        je      clip_parent_text        ;  if no more siblings
        call    read_window_struct

        cmp     dh,w_origin_y[si]       ; check for overlapping windows
        jb      cbt1                    ;  if no overlap
        cmp     dh,w_corner_y[si]
        jae     cbt1                    ;  if no overlap
        cmp     dl,w_corner_x[si]
        jae     cbt1                    ;  if no overlap

        mov     ax,cx
        add     al,dl
        cmp     al,w_origin_x[si]
        jb      cbt1                    ;  if no overlap
        cmp     dl,w_origin_x[si]
        jae     cbt3                    ;  if left overlap
        cmp     al,w_corner_x[si]
        jbe     cbt2                    ;  if right overlap

        pushm   ax,bx,cx,dx,di,si,ds    ;  else middle overlap
        push    ct_offset[bp+di]        ;   save current state and process
        call    cbt3                    ;   exposed right side line
        popm    ct_offset[bp+di]        ;   fragment, then restore and
        popm    ds,si,di,dx,cx,bx,ax    ;   process exposed left side line
                                        ;   fragment
cbt2:   sub     al,w_origin_x[si]       ; here for right overlap
        sub     cx,ax
        jmp     cbt1

cbt3:   mov     al,w_corner_x[si]       ; here for left overlap
```

continued

Intelligent Four Channel

Serial Communications Controller



- 8031 microcontroller
- 82530 SCCs
- 32K memory
- RS-232
- RJ-45 connectors
- DOS device driver
- FCC Class A approved
- Excellent documentation
- BASIC, C and Pascal software source examples
- **$495 complete** with all cables, software, & extensive documentation

Designed, Manufactured, Sold and Serviced by:

Rtech Inc.
Voice . . . (612) 345-4555
FAX . . . . (612) 345-5514
Modem . . (612) 345-4656

907 N. 6th St., Lake City, MN 55041

Circle **36** on Reader Service Card

Total control with *LMI FORTH*™

For Programming Professionals: an expanding family of compatible, high-performance compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics,floating point, native code generation

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761

Circle **63** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        sub     al,dl
        jbe     cbt4                    ;  if complete overlap
        add     dl,al                   ;  set new X coordinate DL=corner_x
        sub     cx,ax                   ;  adjust byte count and text offset
        add     ct_offset[bp+di],ax
        jmp     cbt1

cbt4:   ret
clip_sibling_text endp


;;      clip window text
;
;       entry   BX      windowId (start with active window)
;               CX      byte count
;               DL      column
;               DH      row
;               BP+DI   local variables
;       uses    AX,BX,CX,DX,SI,DS,DI
;
clip_window_text proc
        call    read_window_struct
        jc      cwt1                    ;  if bad windowId
        add     dl,w_origin_x[si]
        jc      cwt1                    ;  if outside window limits
        add     dh,w_origin_y[si]
        jc      cwt1                    ;  if outside window limits

        cmp     dh,w_corner_y[si]
        jae     cwt1                    ;  if below window
        mov     al,w_corner_x[si]
        cmp     dl,al
        jae     cwt1                    ;  if right of window

        sub     al,dl
        mov     ah,0
        cmp     ax,cx
        jae     clip_sibling_text       ;  if line fits in parent window
        mov     cx,ax                   ;  else clip end of line
        jmp     clip_sibling_text

cwt1:   ret
clip_window_text endp


;;      create window
;
;       entry   DS:SI   request packet
;       uses    AX,BX,CX,DX,SI,DS
;
create_window proc
        mov     bx,WReq_WindowId[si]    ;  (get parent window)
        mov     cx,WReq_y_x[si]
        mov     dx,WReq_height_width[si]
        call    create_window_primitive
        jc      cwd1                    ;  if no storage or bad coordinate

        call    reply_packet_read
        mov     WRep_code[si],W_Reply
        mov     WRep_windowId[si],ax
        jmp     transmit_window_reply

cwd1:   call    reply_packet_read
        mov     WRep_code[si],W_Error
        mov     WRep_error_code[si],ax
        jmp     transmit_window_reply
create_window endp


;;      create window primitive
;
;       entry   BX      parent window
;               CL      left most column (parent relative)
;               CH      top most row (parent relative)
;               DL      width
;               DH      height
;       exit    AX      windowId or error code
;               Cf      if no storage or bad coordinate
;       uses    SI,DS
;
create_window_primitive proc
        add     dh,ch                   ; corner is lower right y+1 x+1
        jc      cwp1                    ;  if vertical overflow
        add     dl,cl
        jc      cwp1                    ;  if horizontal overflow

        call    alloc_window_struct
        jc      cwp2                    ;  if no storage

        mov     w_parent[si],bx
        mov     w_origin[si],cx
        mov     w_corner[si],dx
        mov     w_child[si],NULL_WINDOW
        mov     w_sibling[si],NULL_WINDOW
        jmp     link_new_sibling

cwp1:   mov     ax,BadValue
        ret
cwp2:   mov     ax,BadAlloc
        ret
create_window_primitive endp


;;      destroy window
;
destroy_window proc
        ret
destroy_window endp


;;      link new sibling
;
;       entry   AX      new windowId
;               BX      parent windowId
;       uses    SI,DS
;
link_new_sibling proc
        push    bx
        cmp     bx,NULL_WINDOW
        je      lns3                    ; if child of root window

        call    read_window_struct      ; else child of normal window
        mov     bx,w_child[si]
        cmp     bx,NULL_WINDOW
        je      lns2                    ; if first child of this window
lns1:   call    read_window_struct      ; else find youngest sibling
        mov     bx,w_sibling[si]
        cmp     bx,NULL_WINDOW
        jne     lns1
        mov     w_sibling[si],ax        ; and link new sibling
        jmp     lns4

lns2:   mov     w_child[si],ax
        jmp     lns4

lns3:   mov     bx,root_child[bp]       ; set first generation child
        cmp     bx,NULL_WINDOW
        jne     lns1                    ;  if not the first child of root
        mov     root_child[bp],ax

lns4:   pop     bx
        ret
link_new_sibling endp


;;      poll server
;
;       uses    AX
;
poll_server proc
        call    save_most
        mov     cx,SIZEOF_WREQ
        push    ss
        pop     ds
        lea     si,WReq_s[bp]
        call    receive_window_request
        jc      psv1                    ;  if no request pending
        call    process_server_request
psv1:   ret
poll_server endp


;;      process server request
;
;       entry   DS:SI   request packet
;       exit
;       uses    AX
;
process_server_request proc
        mov     al,WReq_code[si]
        cmp     al,W_CreateWindow
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

windowing commands from them. It only supports **WCreateWindow** and **WDrawText**. **WCreateWindow** is relatively straightforward, allocating a window structure and filling it with the parent handle and window coordinates. It then links the window into the parent's sibling list.

The most complex function in the window system is **WDrawText**. Efficient support for overlapping hierarchical windows was a challenge. Rather than process one character at a time, **WDrawText** handles an entire line. Text output begins in **window_text**. It allocates stack space for local variables, receives the display string from the network, and passes control to **clip_window_text**. **Clip_window_text** adds the window coordinates to the cursor position and clips the display string to fit within the window boundaries. If the cursor position is outside the window, it ignores the display operation and returns to the calling program.

Clip_sibling_text receives control from **clip_window_text**. It checks all siblings of higher priority for overlap. If a sibling window completely obscures the string, nothing displays. If the sibling obscures either end of the line, the procedure deletes the hidden line fragment by adjusting the byte count and the text pointer. If the overlapping window cuts the line in two, **clip_sibling_text** processes the two halves independently.

Clip_parent_text chases the parent list and adjusts the cursor position according to the relative position of each parent window. If the string falls outside a window, it is clipped.

Clip_aunt_text is quite similar to **clip_sibling_text**, except the siblings are top-level windows. When clipping is complete, the unobscured string passes to the next layer via **write_video_memory**, which is the interface to video adapter memory. It clips the text to fit within the screen and copies it to the adapter.

Since WSERVER has clipping logic, it may seem inefficient to duplicate clipping logic in WVIDEO. The purpose is to hide knowledge of screen size from WSERVER in a technique called "information hiding." Because WVIDEO is responsible for access to video memory, it is the only layer that can keep information on screen size. Object-oriented languages have specific constructs to support information hiding or "private data." 80x86 assemblers do not, but that does not prevent one from programming that way by convention.

WTEST.ASM is a test program for the windowing system which exercises many possible overlapping window conditions. The code is not pretty, but that's typical of test programs. A more powerful version of this windowing system is available on the code disk for this issue. You can also download it from my BBS at (213) 559-1449.

MORE ON TASM 2.0

I have the new version of Borland's assembler. So far, it looks very good. TASM has an OPTASM-like multipass mode that generates the correct code for a sequence of instructions with forward references.

Unfortunately, the default mode is broken one-pass mode. You must use the switches "/m/jjumps" to enable multi-pass operation. The manual mentions a configuration file, but the index lacks an entry for "configuration."

TASM's multi-pass jump optimizer also has a minor flaw, failing to shrink the following jump to two-byte form. I'm surprised Borland didn't test this:

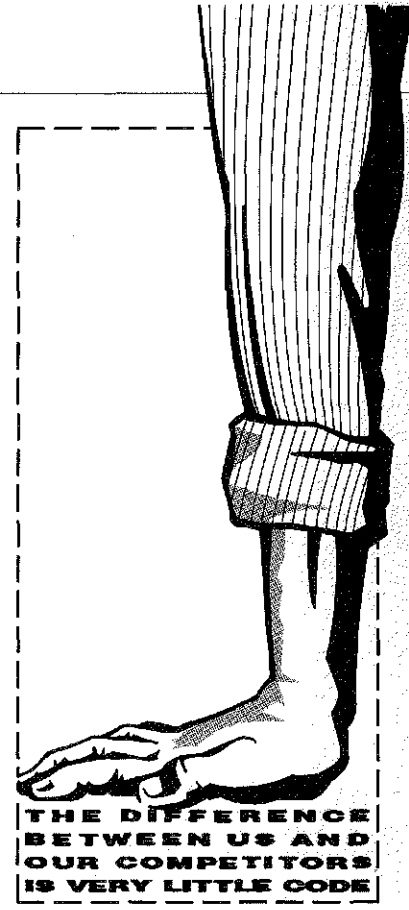
```
        .model  small
        .code
        jmp     foo
        db      127 dup(90h)
foo:    nop
        end
```

TASM performance is good. It is still slower than OPTASM but faster than MASM. TLINK now offers support for the *_edata* and *_end* symbols.

MORE X INFO

You can find more information on X in the following books.

- *X Window System: C Library and Protocol Reference* by Scheifler, Gettys, and Newman.
- *X Window Applications Programming* by Johnson and Reichard.
- *Introduction to the X Window System* by Jones.
- O'Reilly and Associates publishes six volumes of X manuals.



Magic TSR Toolkit

Write complete memory resident programs that require as little as 2K of RAM by swapping to EMS or disk. No secrets: complete assembly language source code and tutorial provided

Quantasm Power Lib

- Over 400 fast, compact assembly language routines
- Overlapping windows in less than 3K
- Over 100 string and numeric functions
- Menus, date, time, sound and more!

Magic TSR Toolkit . . .$199.95
Power Lib. 99.95
Power Lib w/source . . 299.95
ASMFLOW. 99.95

Shipping & Handling: U.S. $5; Foreign $15
CA res. add sales tax
VISA and MasterCard accepted
30-day money back guarantee

QUANTASM CORPORATION

19855 Stevens Creek Blvd., Suite 154
Cupertino, CA 95014
408/244-6826

Circle 252 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        je      psr1                    ; if window open
        cmp     al,W_DestroyWindow
        je      psr2                    ; if window close
        cmp     al,W_Text
        je      psr3                    ; if text output
        ret                             ; else ignore unknown requests

psr1:   jmp     create_window
psr2:   jmp     destroy_window
psr3:   jmp     window_text
process_server_request endp


;;      read window struct
;
;       entry   BX      windowId
;       exit    DS:SI   window structure
;               Cf      if bad windowId
;
read_window_struct proc
        push    cx
        cmp     bx,WINDOW_MAX
        jae     rws2                    ; if bad windowId
        mov     si,bx
        add     si,si
        add     si,si
        lds     si,w_pointers[bp+si]
        mov     cx,ds
        jcxz    rws2                    ; if bad windowId
rws1:   pop     cx
        ret
rws2:   stc
        jmp     rws1
read_window_struct endp


;;      reply packet read
;
;       exit    DS:SI   reply packet pointer
;               CX      byte count
;
reply_packet_read proc
        push    ss
        pop     ds
        lea     si,WRep_s[bp]
        mov     cx,SIZEOF_WREP
        ret
reply_packet_read endp


;;      window text
;
;       entry   DS:SI   window request packet
;       uses    AX,BX,CX,DX,DI,SI,DS,ES
;
window_text proc
        sub     sp,size ct_locals
        mov     di,sp
        sub     di,bp

        mov     ax,WReq_WindowId[si]    ; set local variables
        mov     ct_windowId[bp+di],ax

        mov     ax,WReq_attribute[si]
        mov     ct_attribute[bp+di],ax

        mov     cx,WReq_count[si]       ; receive window text
        lea     si,WReq_text[si]
        call    receive_window_request
        jc      wdt1                    ; if internal error

        mov     ct_offset[bp+di],si     ; set text pointer
        mov     ct_segment[bp+di],ds

        call    clip_window_text        ; clip text & write to video memory

wdt1:   lea     sp,[bp+di+size ct_locals]
        ret
window_text endp

        end
```

Listing 4

```
        title   miscellaneous procedures

        include asm.inc

        .code
        extn    calloc_raw

;;      calloc read
;
;       entry   CX      requested byte count
;       exit    DS:SI   pointer to zeroed storage bucket
;               Cf      if no storage
;       uses    AX
;
calloc_read proc
        pushm   di,es
        call    calloc_raw
        jc      carl
        push    es
        pop     ds
        mov     si,di
carl:   popm    es,di
        ret
calloc_read endp

        end
```

Listing 5

```
;NOTE: this listing contains only excerpts from WVIDEO.ASM.
;All of the listings are included on code disks and at PJ BBS sites.

        title   wvideo - video interface for window system
        include window.inc
        include asm.inc

        .data?
video_pointer   label   dword
video_offset    dw      ?
video_segment   dw      ?

row_count       db      ?
column_count    db      ?


        .code
        extn    create_window_primitive


;;      convert pixel value
;
;       entry   AX      pixel color value
;       exit    AL      attribute
;
convert_pixel_value proc
        cmp     ax,BLACK_PIXEL
        jne     cpv1
        mov     al,70h
        ret
cpv1:   cmp     ax,WHITE_PIXEL
;       jne
        mov     al,07h
        ret
convert_pixel_value endp


;;      video io
;
;       entry   AX,BX,CX,DX see INT 10 documentation
;       exit    AX,BX,CX,DX see INT 10 documentation
;               Cf      0
;
video_io proc
        push    bp
        int     10h
        pop     bp
        clc
        ret
video_io endp

        end
```

PJ CODE

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 16 and 80.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ PRESSROOM

FAST LINKERS FROM SLR

SLR Systems has released a pair of fast new linkers for DOS and OS/2. DOS-based Optlink/Compress incorporates optional file compression, which yields fast self-extract-to-run EXE files that are 25 to 50% smaller than /EXEPACK, and even smaller than PKZIP files. The executable files created with the compress option need less than 1K of memory to self-extract. Quick self-extract code is fast enough for most applications. SLR offers a royalty-free UNPACK utility that can be shipped with your product to extract to disk in case start-up time is a higher priority than disk space.

Optlink/2 is a Microsoft-compatible segmented linker for generating 16-bit OS/2 EXE and DLL files. It also generates COM, SYS, and EXE files for DOS. The OS/2 definitions files (DEFs) used are compatible with the Microsoft OS/2 linker. Also included is OPTIMP, an import library builder.

The SLR Optlink linkers are the fastest available for both DOS and OS/2. Optlink/Compress and Optlink/2 are priced at $250 each. Upgrades from previous versions of OPTLINK are also available.

SLR Systems, Inc.
1622 N. Main Street
Butler, PA 16001
412-282-7965

TURBO C++ PROFESSIONAL

Borland International has recently released two new C++ development environments for DOS: Turbo C++ and Turbo C++ Professional. Turbo C++ provides a complete AT&T C++ 2.0 native code compiler, as well as offering full compatibility with ANSI C. Turbo C++ includes a new integrated development environment called the Programmer's Platform. This enables programs to break the 640K memory barrier by incorporating Borland's VROOMM (Virtual Runtime Object-Oriented Memory Manager).

VROOMM permits applications to use EMS, extended memory, or disk swap space to create large executable programs. It eliminates the complexity associated with traditional methods of managing overlays by determining the program modules to be held in memory at runtime.

Programmer's Platform provides a new multifile editor with overlapping windows, mouse support, dialog boxes, an intelligent project manager, context-sensitive hypertext help, and Borland's integrated debugger. A transfer option allows the integration of other tools and compilers into Programmer's Platform. Turbo C++ Professional includes all of Turbo C++'s features, along with separate standalone versions of Turbo Debugger, Turbo Assembler, and the new Turbo Profiler.

Turbo Profiler aids programmers in measuring a program's performance and spotting bottlenecks. It provides a histogram of time-critical statistics, including how often each line is called, the time spent in each routine, and how system resources are being used.

Turbo C++ retails at $199.95 and includes four manuals and on-line hypertext help. The Professional package adds five manuals and costs $299.95. Upgrades from Turbo C are available at reduced cost.

Borland International, Inc.
1800 Green Hills Road
Scotts Valley, CA 95066-0001
800-331-0877

ZINC INTERFACE LIBRARY

Zinc Software has announced the Zinc Interface Library. The library is a full-featured, customizable, object-oriented user-interface class library specifically designed for C++. The Zinc Interface library uses the object-oriented features of C++ to provide classes which include a screen display, an event manager, a window manager, a help system, and an error reporting system.

The screen manager supports both text and graphics mode. Graphics support is compatible with the Borland Graphical Interface (BGI). The event manager maintains a linked list of input events such as a keyboard or a mouse. The window manager contains a full-featured set of windows objects including scroll bar, pull-down menu, icons, and more. The help system allows programmers to provide context-sensitive help. The error system gives the application user several options in handling run-time errors.

The complete Zinc Interface Library is priced at $200 and comes with a full 60-day money-back guarantee.

Zinc Software Incorporated
405 South 100 East, Suite 201
Pleasant Grove, UT 84052
801-785-8900

OVERLAY DESIGNER

Coleman Softlabs announces Overlay Designer, a tool for designing overlay structures using Plink86 or RTLink. Overlay Designer is targeted for programming teams building products with more than 50 overlays. It reads the MAP file the linker puts out in order to get the information necessary to build its database. Overlay Designer will then generate an on-screen or a hardcopy diagram of the link and uses graphs, algorithms, and tabulation tools to refine the overlay structure.

Using the on-screen user interface, the developer can receive a report on symbol usage, move code between various overlays and instantaneously get a computed result, or generate tables to show program sizes by file, module, and overlay. Overlay Designer is priced at $1,200 and comes with a large, indexed manual and with on-screen, context-sensitive help.

Coleman Softlabs, Inc.
296 Bay Road
Atherton, CA 94027
415-322-9006

TCP/IP FOR OS/2 AND DOS

FTP Software has released PC/TCP for OS/2, an implementation of TCP/IP internetworking protocols. This product provides TCP/IP networking for all versions of OS/2, including IBM's Extended Edition. PC/TCP makes it possible for OS/2-based PCs to communicate with other PCs as well as with different types of computers running a variety of operating systems. In addition, PC/TCP provides a comprehensive set of TCP/IP utilities for OS/2 machines and uses Presentation Manager for multitasking and windowing.

PC/TCP has been designed to pro-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

vide a high degree of hardware independence. The OS/2 version operates on any network interface card with an NDIS (Network Driver Interface Specification) driver, including Ethernet and Token Ring Products from 3Com, Western Digital, Excelan, and Interlan. NDIS drivers for these cards are included in the distribution package.

FTP Software has also announced that the PC/TCP Development Kit will be available sometime in September. This kit provides direct access to the transport, network, and link layers of PC/TCP. It includes a Berkeley Socket emulation library which allows direct access to TCP, UDP, IP, name resolvers, and VT100 and H19 terminal emulators. In addition, it supports all memory models of Microsoft C.

FTP also has plans to add two more OS/2 internetworking products: Interdrive, a Network File Server (NFS) client implementation, and NetBIOS, an interface between applications and a network. These options are already available for the DOS version.

PC/TCP for OS/2 is priced at $575. The Development Kit costs $500, Interdrive is $100, and NetBIOS costs $80.

FTP Software, Inc.
26 Princess Street
Wakefield, MA 01880
617-246-0900

NUMBER SMASHER-i860

MicroWay has announced that its Number Smasher-i860 coprocessor board and NDP-i860 compilers are scheduled to ship in September. Their i860 hardware and software products provide PC users with an integrated, high performance i860 solution. The i860-powered Number Smasher delivers up to 80 million single precision floating point operations per second (MFLOPS) at 40 MHz and produces over 10 double precision Linpack megaflops. With the NDP-i860 compilers, source code from mainframes, 80386/486s, or PCs can easily be ported to i860 and run on MicroWay's XTEND-i860 environment under DOS, UNIX, or XENIX.

Number Smasher-i860 is a 33- or 40-Mhz i860 coprocessor board for the ISA bus that comes standard with 8 megabytes of 64-bit, four-way interleaved DRAM. A 32-megabyte option is available as well. The board runs in 80236, 80386, and 80486-based micros.

The 33 Mhz Number Smasher-i860 is $5,995; the 40 Mhz version is $7,200. Both versions include 8 megabytes of memory, an NDP-i860 compiler, diagnostic software, and a one-year warranty. Additional NDP i860 compilers sell for $1995 each, and the optional vectorizer is $995. All of the compilers include one year of free updates.

MicroWay
P.O. Box 79
Kingston, MA 02364
508-746-7341

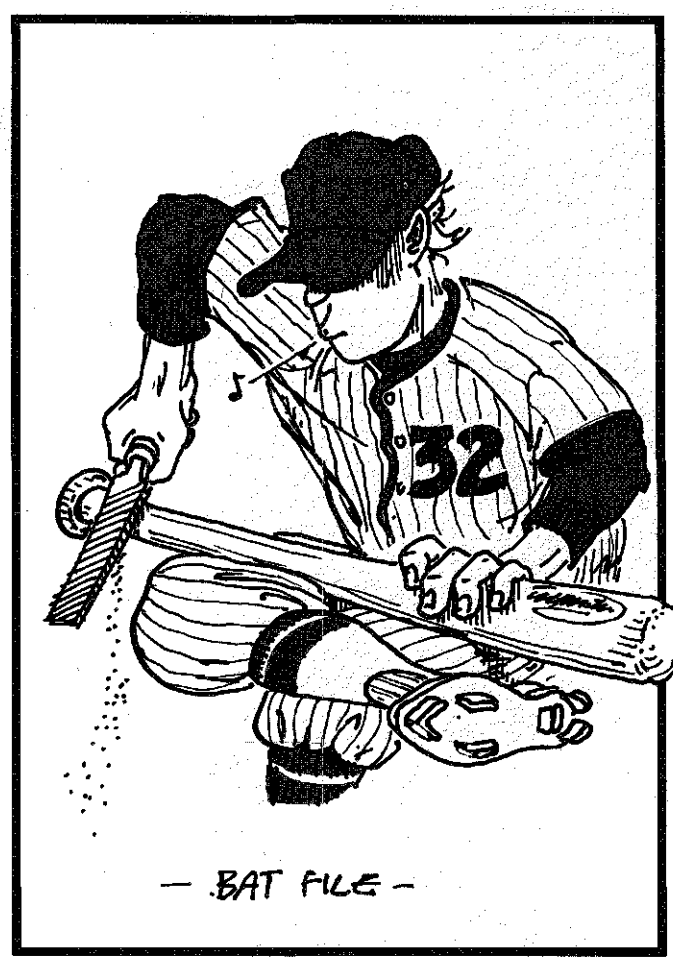
The Programmer's Corner
— A Source Code Repository —

If you are reading this magazine, then you appreciate the art of programming. One of the most important tools for a software craftsman is example code. To meet this need, we provide **more than 9000 files** (over one gigabyte) of shareware and public domain source code and programs. We cover all popular languages including C, Pascal, and Basic. TPC even has hard to find source code such as OS/2 PM C source, C++ source, and Turbo Pascal 5.5 OOP source. Source code from this issue of *Programmer's Journal* is also available. New code is added daily, all in the interest of helping the professional programmer.

So call today: (301) 596-1180 300-9600 Baud
(301) 997-3744
Only $50 for a one year subscription

— TPC uses FastComm 9600 V.32 Modems —

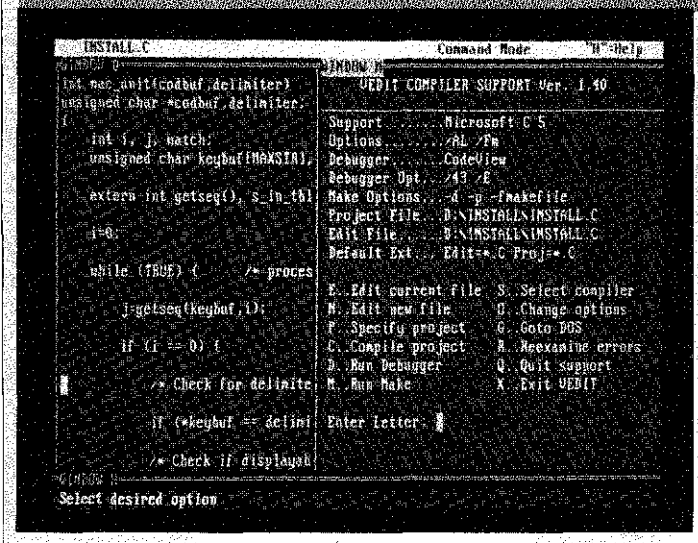
TPC
The Programmer's Corner
5430 F Lynx Lane, Suite 224
Columbia, MD 21044



Circle 54 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Finally, a fast, powerful text editor that integrates your favorite programming tools and uses no memory!

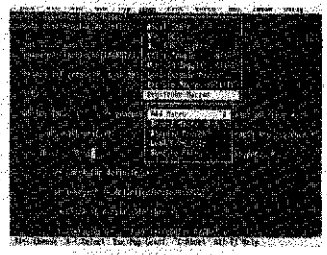


VEDIT PLUS

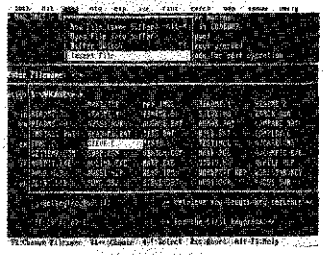
FREE Fully Functional Evaluation Copy Call 1-800-45-VEDIT

- **Mouse Support**
- **Pull-down menus**
- **Columnar blocks**
- **1000 Level Undo**
- **Regular expressions**
- **Small 70K size, fast**
- **Emulate Wordstar, Word Perfect, Brief, vi, others**
- **Edit text and binary files of any size and line length**
- **Powerful macro programming language for instant "off-the-cuff" macros**

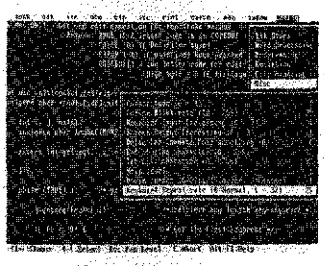
DOS XENIX UNIX QNX FlexOS



An intuitive user interface with pull down menus, hot keys, mouse support and context sensitive help make VEDIT easy to use, easy to learn.



Point and shoot file selection makes it a snap to edit new files, merge and split files, run macros and much more. Select files with the cursor keys or the mouse.



Just about everything about VEDIT is configurable, from the keyboard layout, to the screen colors, to the way control characters, tabs and the end of lines are displayed. Configure VEDIT with easy to use menus.

The new VEDIT family of text editors offers stunning performance, versatility and ease of use. Completely written in assembly language, they are small and lightning fast. (3 to 30 times faster than other editors on large files where speed really counts.) Edit text and binary files of any size, even 100+ megabytes. Installation is trivial; VEDIT.EXE and an optional help file are all you need - no overlays, no configuration files, no environment variables.

For programmers, the new compiler support in VEDIT and VEDIT PLUS is a breakthrough. Run not only popular compilers, but debuggers and your favorite tools from within the editor. When shelling to DOS, VEDIT swaps itself and any desired TSRs out of memory to give you more memory than when you entered VEDIT. Only VEDIT offers you the advantages of a powerful editor without giving up the convenience od an integrated environment.

Call for your free, fully functional, evaluation copy today. See why VEDIT has been the choice of 100,000 programmers, writers and engineers since 1980.

VEDIT Jr. - Unmatched performance for only $29.

All VEDIT editors include a pull-down menu system with "hot keys", context sensitive help, pop-up status and ASCII table, a configurable keyboard layout and flexible, unlimited keystroke macros. Perform block operations by character, line, file or column. Undo up to 1000 keystrokes - keystroke by keystroke, line by line, or deletion by deletion. Automatic indent, block indent and parentheses matching speed program development. Word wrap, paragraph formatting, justification, centering, adjustable margins and printing for word processing. Run DOS programs.

VEDIT - A best value at only $69.

VEDIT can simultaneously edit up to 36 files and split the screen into windows. Search/replace with regular expressions. The most integrated compiler support available. Run VEDIT PLUS macros.

VEDIT PLUS - Ultimate programmer's tool for only $185.

VEDIT PLUS adds the most powerful macro programming language of any editor. It eliminates repetitive editing tasks and permits creating your own editing functions. The macro language includes testing, branching, looping, user prompts, keyboard input, string and numeric variables, complete control over windows plus access to hardware interrupts, memory and I/O ports. Source level macro debugging with breakpoints and tracing.

VEDIT PLUS - $185 for DOS, $285 for UNIX/XENIX. 30 Day Money-back guarantee. Discount pricing for multiple users, schools and OEMs. VEDIT Jr. site licenses start at only $250.

BENCHMARKS IN 3 MEG FILE

	VEDIT	BRIEF	Sage	QEdit
Simple search	57 sec	1:17 min	28 sec	Cannot
Save and continue	52 sec	3:52 min	1:47 min	Cannot
Load, modify, save, exit	21 sec	49 sec	1:38 min	Cannot
Block-column copy (40x200)	2 sec	30sec	2 sec	2 sec
Delete one column in file	**9:58 min**	**1:50 hour**	**1:03 hour**	Cannot
60,000 replacements	**3:18 min**	**1:44 hour**	**1:32 hour**	Cannot

VEDIT is a registered trademark of Greenview Data, Inc. BRIEF is a trademark of UnderWare, Inc. Sage Profession Editor is a trademark of Sage Software Inc. QEdit is a trademark of SemWare.

* DOS version also supports Concurrent DOS, DESQview, Microsoft Windows, PC-MOS/386 and most networks. Special CRT terminal version runs over serial lines and modems.

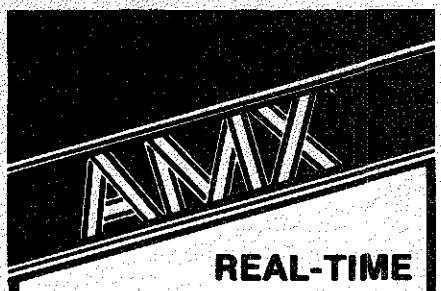
* Free evaluation disk is fully functional and can edit small files.

Greenview Data

P.O. Box 1586, Ann Arbor, MI 48106
(313) 996-1299, Fax (313) 996-1308

Circle **56** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



REAL-TIME MULTITASKING KERNEL

8086/88, 80x86/88 80386
Z80, 64180, 8080/85 68000/10/20

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
  - — task create/delete
  - — task priorities
  - — memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- InSight™ Debugging Tool

THE BEST

Join over 600 developers such as IBM®, Xerox, Hewlett Packard, Hayes, Hughes Aircraft and NASA.

CHOOSE AMX

The best low-cost, high-performance real-time multitasking system available today.

No Royalties
Source Code Included

Demo Disk	$25 US	*Call for prices for other processors.*
Manual only	$75 US	
AMX 86	$3000 US	

(Shipping/handling extra)

IBM is a registered trademark of IBM Corp.
Z80 is a trademark of Zilog, Inc.
AMX, AMX 86, inSight are trademarks of KADAK Products Ltd.

KADAK Products Ltd.
206-1847 West Broadway
Vancouver, B.C., Canada
V6J 1Y5
Telephone: (604) 734-2796
Fax: (604) 734-8114

SMALLTALK-80 FOR WINDOWS

ParcPlace Systems plans to release Objectworks for Smalltalk-80, an object-oriented programming system for Microsoft Windows 3.0, in the fall of this year. This new product will bring workstation performance to the PC and allow programmers to port graphical user interface (GUI) programs across PCs and workstations.

Code optimization for Windows 3.0 will result in performance improvements to Objectworks for Smalltalk-80. Smalltalk-80/Windows takes advantage of Windows 3.0's memory management and runs in the 80386 enhanced mode, eliminating the 640k program size barrier. With access to virtual memory, large programs can be run on systems with minimal memory.

For example, up to a 16-megabyte Objectworks application can run in as little as 4 megabytes of physical memory. Objectworks runs in the 32-bit linear mode using the DOS Protected Mode Interface, providing additional performance advantages.

Objectworks offers a complete software development environment, an object-oriented development language with dynamic compilation, and an extensive class library of more than 300 reusable software modules. Objectworks for Smalltalk-80/Windows 3.0 is priced at $595.

ParcPlace Systems
1550 Plymouth Street
Mountain View, CA 94043
800-822-7880

NEW PDC PROLOG

The **Prolog Development Center** (PDC) has announced the availability of PDC Prolog Version 3.2, a significantly updated version of the Turbo Prolog language compiler previously marketed by Borland International. It is available for both DOS and OS/2, and includes a number of new features and enhancements.

The interactive environment is fully customizable, with separate windows for source code editing, tracing, system messages, and dialogs. PDC Prolog supports text mode windowing, BGI graphics, and multiple internal and external databases for Prolog objects (terms) with support for B-tree file indexing, EMS and network file-sharing.

The OS/2 version includes a complete subsystem of named pipe services, allowing two or more programs to exchange Prolog terms even across networks. Also, there is support for totally asynchronous transmission and reception of messages independent of the main program's operation.

The PDC Prolog compiler for DOS is $249, or $79 if upgrading from Turbo Prolog. For OS/2 the price is $599, including the DOS version, or $429 for an upgrade from Turbo Prolog.

Prolog Development Center
568 14th Street NW
Atlanta, GA 30318
404-873-1366

DESQVIEW/X

Quarterdeck has announced that it will incorporate the X Window System into its DESQview environment and into its DESQview API developer tools. DESQview/X is a full implementation of the X Window System for DOS, running X server and clients locally and simultaneously with DOS programs. The system will also provide capability to run DOS and X Window programs remotely over networks on any X Window-based machine.

Network support currently includes Novell, TCP/IP, and other protocols. For the DESQview user on a standalone 80286 or 80386 PC, this technology will bring a 3-D look and feel and an icon-based desktop to DESQview.

DESQview/X will multitask not only DOS, 286, and 386 extended applications, but also local 16-bit and 32-bit X Window applications. OSF Motif and Open Look window managers will be available options to ensure a consistent user interface for those currently running X-based applications on workstations (e.g., DECstations and Suns). DESQview/X will be compatible with EGA, VGA, EVGA, IBM 8514, and DGIS graphic display standards. The X Window System for DESQview is planned for release in late 1990.

Quarterdeck Office Systems
150 Pico Boulevard
Santa Monica, CA 90405
213-392-9851

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Multitasking

Do the multitasking systems you are considering using seem a bit like overkill or need an awful lot of memory? Maybe all you really need is a system which can run more than one DOS program concurrently. If this sounds like your situation, consider MultiDos Plus. Flexible preemptive scheduling of multiple tasks with full access to MS-DOS or PC-DOS. LIM 4.0 support. For more information please call for a free brochure.

Nanosoft

800-678-2141
508-651-0091
FAX 508-655-8860
13 Westfield Road
Natick MA 01760

Circle **109** on Reader Service Card

PRINTER GRAPHICS IN HIGH RESOLUTION!

GRAPHLINK™ Printer Graphics Toolkit

GraphLink™ lets your programs tap the *full potential* of printers to produce dazzling *printed* graphics in *high-resolution!*

- Uses the PRINTER'S resolution, not the SCREEN'S!
- Supports the most popular laser, inkjet, and dot matrix printers
- 80+ routines in a linkable library
- Fully scalable fonts
- Emulates the Borland Graphics Interface - great for BGI programs!
- Works with all screen graphics libraries
- Can *automatically* build the printer image
- Supports Turbo Pascal 4.0 and higher
- Available soon for Microsoft C, Quick C, and Turbo C
- *NO ROYALTIES!*
- Only *$199* (shipping: USA $5, Canada $10, Other $15) (with Source Code only $299)

30 Day No Risk Money-Back Guarantee

GRAPHLINK
The Printer Graphics Solution™

VISITECH SOFTWARE
D5 3807 Ridgewood Court
Pittsburgh, PA 15239 USA
(412) 733-4775

Circle **304** on Reader Service Card

MISOSYS, Inc.

Programmer's Journal downloads are now available from MISOSYS' forum on CompuServe: PCS49

When you don't have to write in stone, don't let your editor weigh you down. You need SAID-86!

Editing was never so easy!

SAID-86 is a fast, flexible, full screen text editor for PC's. It is perfect for editing batch files, program listings, README files, CONFIG.SYS files, and anything you now do with EDLIN or the non-document mode of a word processor. Why struggle with huge editors; when all is said and done, SAID-86 will be your text editor of choice!

Check out this list of features

- ✓ WordStar-like editing commands are easy to use
- ✓ Pull-down menu system for commanding SAID-86
- ✓ Supports nine editing buffers with automatic swap to disk
- ✓ Supports up to 30 user-defined macros; 255 characters each
- ✓ Undelete the last nine deleted lines can save your bacon
- ✓ MOUSE support with automatic recognition
- ✓ HELP facility; shell to invoke DOS commands from SAID-86
- ✓ SAID-86 can expand or contract TABs

SAID-86 is reasonably priced at just $29.95 + $2S&H

DJ10 Jumbo	$295 +$5S&H
AB10 Tape Adaptor	$75 +$3S&H
KE10 External Adaptor	$110+$5S&H
DC2000 tape	$22.50

M/C & VISA. S&H are U.S. only.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS
(US&Canada)
or 703-450-4181

Now get the clearest view of your disk drive and memory contents with our DED86™

Powerful features in Version 2!

When you need to travel through your disk drive, why settle for a tool that isolates you from the road? DED86 gives you the direct controls you need to explore your disk. It's a full-screen sector-oriented disk/file editor and a page-oriented memory editor. When you want to "unerase" erased files, DED86's flexible KEEP facility does the job without you fussing over FATs.

Check out this list of features

- ✓ Look by cylinder/head/sector, sector or cluster; even reserved/hidden sectors
- ✓ Scan free clusters to search for erased data ✓ Jump about subdirectories
- ✓ Keep sectors & clusters for writing to a file; Great for partial file recovery
- ✓ Edit bytes in hexadecimal or ASCII, zap in 0s; Undo last edit!
- ✓ Search your disk drive, a file, or memory for ASCII or hexadecimal strings
- ✓ Touch a directory file entry with your date and/or time
- ✓ Obtain complete disk statistics in one screen; includes usage data
- ✓ Alter file attributes: archive, system, hidden, read
- ✓ Save/Restore sectors to/from auxiliary buffers; for moving data around
- ✓ DOS subshell available while using DED86 lets you shell to other programs
- ✓ Handles 5.25" & 3.5" drives & RAM disks; even large partition drives

DED86 is easy on your pocketbook; only $29.95 + $3S&H

Fujitsu Floppy Disk Drives:	5.25" 360K 1/2 height	$75+$4S&H
	3.5" 720K in 5.25" frame	$85+$3S&H
	3.5" 1.44M in 5.25" frame	$90+$3S&H

Circle **121** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

FIGHT PIRACY & PROTECT YOUR PROGRAM $$$'s!

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not choose one of these "proven leaders":

- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Security
- **EVERKEY** Hardware "Key" Software Security

For IBM and Compatibles.
30 day money back guarantee.
Free info and demo disk available.

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085
(800) 227-0644 (816) 776-2700
FAX (816) 776-8398

Circle 51 on Reader Service Card

HUGE Professional Shareware Libraries

Comprehensive, indexed collections of all known Public Domain and Shareware utilities for PC professionals, updated quarterly with the latest products. Includes extensive disk based or printed directories with multiple search options. Each library designed specifically with the PC professional in mind. Computer Language called our dBase product "possibly the most productive $100 you'll ever spend". VISA, MC, AmEx, POs, COD.

Product	Disks/Files	Price
dBase/dBase Compilers	66+/1300+	$99.50
AutoCAD	17+/570+	$99.50
C	28+/275+	$49.50
Turbo Pascal	23+/270+	$49.50
Ventura DTP	25+/200+	$49.50
Lotus 123	17+/200+	$49.50
Assembler	14+/200+	$49.50
Games	35+/220+	$49.50
Novell Netware	24+/200+	$49.50
WordPerfect	30+/160+	$49.50
Utilities for DOS Gurus	26+/230+	$49.50

EMS, 4505 Buckhurst Ct.
Olney, MD 20832 USA
(301) 924-3594 Fax: (301) 963-2708
France (1)47763872 Germany 0911/338844
Benelux 31-20-5753019 Singapore 4671880

Circle 353 on Reader Service Card

EZ-COPY PLUS

Flawless Disks Fast!

•2X+ faster than DOS •Read diskette once, then.. •Quickly & accurately mass duplicate 5.25" & 3.5" disks (360K, 720K, 1.2M, 1.44M) on your own PC/XT/AT/etc. © •Supports up to 4 drives with automatic start and switching. •Formats, copies, verifies, etc. • Partial copies too. •Even faster with pre-formatted disks. •Serializes disks and/or labels, in 1 smooth operation. •Save disk images to HD, more... •Replaces hardware worth $1000s.

Only $495 +s/h NCP for upto 10 users.

S&H: $6 USA48; $10 N.Amer; $22 Foreign.
EZX Publ., 917 Oakgrove Dr., Ste PJ990, Houston, TX 77058 •FAX: 1-713-280-0525

INFO 1-713-280-9900
V/MC/AX Orders & Brochures Toll Free:
1 • 800 • 359 • 953 • 9 xPJ990

Circle 352 on Reader Service Card

FREE CATALOG!

LATEST AWARD BIOS
PC/XT, 286, 386

COMPILED BASIC TOOLS
Libraries, Utilities
for QuickBASIC 4.5

HARDWARE UPGRADES
for PCs & PC/XTs

(800) 423-3400 - (412) 782-0384
KOMPUTERWERK INC.
851 Parkview Blvd.
Pittsburgh, PA 15215

Circle 155 on Reader Service Card

RELATIONAL DATABASE LIBRARY

For TurboC, Microsoft C on DOS systems. The 114 functions in **"c_wndw"** and **"c_ndx"** libraries support:

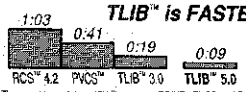
- Multiple B-tree index files
- Stacked, overlapping windows
- dBase .DBF and .NDX file formats
- Multi-level file relationships
- Pull-down and pop-up menus
- Hashed random file access
- Data dictionary field access

Libraries $99.50; Source $199.50; VISA/MC

Marietta Systems, Inc.
P.O. Box 71506
Marietta, GA 30007
(404) 565-1560

Circle 223 on Reader Service Card

TLIB™ is FASTEST!



Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. MKS RCS 4.2 and TLIB 5.0 are newer.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**

"TLIB is a great system" J. Vallino, **PC Tech J**

- **Full-Featured Version Control for Software Professionals.** Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, etc.. Integrates with **Opus™ MAKE** & **Slick™ MAKE**.

MS-DOS $139, OS/2 $195 + shipping Visa/MC
5 station LAN license $419 (OS/2 $595), call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 **(919) 233-8128**

Circle 288 on Reader Service Card

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use 'subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. Only **$59.95**

ORDER TODAY!

Wenham Software Co.
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Circle 320 on Reader Service Card

FREE TRIAL

Pinnacle File Manager

We guarantee immediate productivity gains from using our C file manager library. Simple table-and-column view of data with advanced B-Tree indexing and hi-speed cache frees your brain to solve application-specific problems.

Vermont Database Corp.
RR3 Box 6516
Stowe, VT 05672
800-822-4437

Circle 328 on Reader Service Card

WRITE SOURCE CODE IN ANY MS LANGUAGE AND LINK TO ONE WINDOWING LIBRARY

STOP PLAYING 'MUSICAL' LIBRARIES ONE VERSION OF MXW LINKS TO ALL!
(Quick Pascal/COBOL excluded)

- MULTIPLE Window Designs
- SCREEN Paging, SPLIT Windowing
- POP DIRECTORIES in a Window
- In-System MENU GENERATION
- MOVE WINDOWS Rapidly
- Powerful SCREEN MANAGEMENT
- TOP SCREEN Menuing System
- BAR MENUING Systems - Any Row

And a Great Deal More . . .

$69.95 + shipping and handling
Complete Documentation on Disk.
COD, Check or Money Order Accepted
Call or Write for Details and Demo Disk

MTD Software Dev. LTD
516-434-1979
21 Ralph Ave, N. Brentwood, NY 11717

Circle 312 on Reader Service Card

dBASE FILE ACCESS FROM C

Code Base 4 is a library of C routines which gives complete dBASE and Clipper functionality and file compatibility. The routines correspond directly to dBASE commands. Create multi-user applications using pulldown menus, popup windows and data entry.

Code Base 4 supports DOS, UNIX, OS/2 and Microsoft Windows. $295 with Source! FREE DEMO.

Sequiter Software Inc.
P.O. Box 5659, Station L
Edmonton, Alberta Canada T6C 4G1
403-448-0313
Fax: 403-448-0315

Circle 269 on Reader Service Card

The $25 Network

Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs, PS/2s
- Uses serial ports and 5 wire cable
- Runs at 115K baud, up to 90 feet
- Transfer 8500 bytes per second (ATs)
- Runs in background, totally transparent
- Share any device, any file, any time
- Needs only 14K of ram
- Replace all file transfer software
- $25 per network, NOT PER NODE!
- OVER 15,000 SOLD WORLD WIDE

Skeptical? We make believers!

imodes Information Modes
Drawer F, Denton, TX 76202
817-387-3339 Technical
1-800-628-7992 Orders

Circle 348 on Reader Service Card

OPT-TECH SORT/MERGE

Extremely fast Sort / Merge / Select utility. Run as an MS-DOS command or CALL as a subroutine.
Supports most languages and filetypes including Btrieve and dBase. Unlimited filesizes, multiple keys and much more!

MS-DOS **$149.**
OS/2, UNIX **$249.**

Opt-Tech Data Processing
P. O. Box 678
Zephyr Cove, NV 89448
(702) 588-3737

Circle 16 on Reader Service Card

Offer your product directly to Professional Programmers:

Placing an ad in **PJ BILLBOARD** is an easy, low-cost way to reach potential buyers. Regular listings include a headline, company information, and approximately 400 characters for your message. Display ads are 1 5/8" wide by 2 3/8" high. For more information or to reserve a space, call our ad department at **503-747-0800.**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DO IT YOURSELF!

XenoCopy-PC

PC-DOS program lets your PC *Read/Write/Format* *over 350 formats* **$79.95**

XenoFont

high quality text screen printouts

$49.95

+ $5.00 S/H + Tax if CA.

XenoSoft • (415) 644-9366
2210 Sixth Street,
Berkeley, CA 94710

Circle 123 on Reader Service Card

sPORTt

serial PORT tester

sPORTt is a serial port tester for people who don't want to be data communications experts, but use or write software for serial devices, such as plotters, mice, digitizers, modems, serial printers, or computer to computer links.

- displays all data going in and out in ASCII, decimal, hex, binary
- lets you choose handshaking lines
- sends/receives disk files
- automatically runs port diagnostics

Order today only $89.95!

WHS
3037 Grass Valley Hwy #8201
Auburn, CA 95603
916-885-2480

Circle 329 on Reader Service Card

PC TIMER TOOLS

New Version!

PCHRT is the definitive answer to execution profiling, precision timing and timer tick interrupt management in a PC/MSDOS environment. Over 50 functions manage multiple microsecond timers, generate precision delays, insert timers in most any interrupt, and run the timer tick interrupt at a user specified rate. Utilities to time execution and profile interrupts of any .EXE. Supports TC, TP, and MSC. Full library source included. $49.95 ppd USA, elsewhere add $4.00. VISA/MC.

RYLE DESIGN
PO Box 22, Mt. Pleasant MI 48804
517-773-0587

Circle 210 on Reader Service Card

IS 640K JUST NOT ENOUGH?

The **ATLAS MEMORY BOARD** adds 64K of DOS USABLE memory to the base 640K, making the total CONVENTIONAL memory 720,896 bytes. It does not use any memory resident drivers, and is compatible with all known applications. The same board works in any PC, XT, AT or 386 compatible, with monochrome graphics or CGA video. Memory added is NOT extended or expanded memory. The Atlas memory board has been certified by both NOVELL and 10 Net. Most orders shipped same day.

ATLAS MEMORY BOARD
15 Day Free Evaluation **$149.00** Qty 1 Quantity Discounts Available

M/C 1-800-535-6712 VISA
9AM - 9PM MON-SAT
NATIONAL DISCOUNT COMPUTERS
702-459-0049 NV Residents & Tech Support

Circle 107 on Reader Service Card

QUICKBASIC/PDS 7/NETWORK

B+ TREE FILE INDEXING

Maintain B+ Tree file indexes in your QuickBASIC/PDS 7/NETWORK programs with INDEX MANAGER. Access files randomly by full or partial key, or sorted forward or backward. One module performs all functions, making IM very easy to use. Only indexes managed! You retain full control over your data. Assembler code and cache buffers make IM very fast. Includes sample programs and User's Guide. Only $59 (+tax in CA). PDS 7 version $99. Network version $250.

CDP Consultants
1700 Circo Del Cielo Drive
El Cajon, CA 92020
619-440-6482

Circle 335 on Reader Service Card

Developer's Assistant

A comprehensive system providing the essentials all developers need. Database to maintain users and dealers. Uses multiple indexes for fast lookups. Lots of reports. Utilities to embed serial numbers in your program's sign on screen. A label designer and more. Professional directory of packagers, duplicators, associations, lawyers, etc. Directory of Editors for Product Releases.

Plus **FREE** use of a 900 number to support your product and earn money by the minute. $69.00 (+tax in CA)

S.L.A.M. Software
2523 El Portal Dr. Suite 101-25
San Pable, CA 94806
512-441-2615

Circle 12 on Reader Service Card

UPPER DECK FORTH

$49

Based on the Forth-83 Standard. Fully segmented architecture. Fast compile, no link required. Uses normal DOS files. Includes editor, assembler, decompiler and debugger. Turnkey application support. For PC's and compatibles, 256K, DOS 2.0 or later. Add $3 for shipping and handling (outside USA $15). CA residents add sales tax.

UPPER DECK SYSTEMS
P.O. Box 263342
Escondido, CA 92026
(619) 741-1075

Circle 24 on Reader Service Card

The EDitor

Powerful **UNIX**-style programmer's editor. For use on IBM or compatible. Comprehensive manual included. A solid value at only $74.95 (US).

Available from:

Blue Jay Systems
PO Box 608
Mount Vision, NY 13810
607 293-8007

Circle 11 on Reader Service Card

GRAF/DRIVE PLUS

Turbo C/Pascal Graphics Drivers for Printers and Plotters

LaserJet–Epson–HP7470. Just plug our BGI-compatible drivers into Turbo's *initgraph*.

- Port your screen graphics program to hard copy devices with only minor changes.
- Get the full resolution of the output device, not a screen dump.
- Not TSR, Max 16K.

Personal $149. No-royalty developer $299. 30-day m/b guarantee.

FLEMING SOFTWARE
PO Box 528 Oakton, VA 22124
(703) 591-6451

Circle 178 on Reader Service Card

SHAREWARE AUTHORS

Give your programs the chance they deserve! With Megapost, your program is uploaded to CompuServe, GEnie, Delphi, BIX, and EXEC-PC. The low fee is probably less than you would pay just to open accounts with all of these services. Included are one half-price update and notification of the downloading procedure and 3-month download count for each service. 15% discount for ASP members.

Write for a free brochure.

Andrew M. Saucci, Jr.
641 Koelbel Ct.
Baldwin, NY 11510-3915

Circle 347 on Reader Service Card

80386 MACRO DISASSEMBLER

MASTERFUL DISASSEMBLER is the worlds first macro disassembler. Interactive by design not after thought. Features:

- User defined macros with arguments
- Auto code/data separation
- Auto commenting and label naming
- Command menus and help screens
- Customizable disassembly options
- Comprehensive, indexed, 100 pg manual
- Fully supports 8086/87 thru 80386/87
- COM/EXE/SYS or other files or memory
- Cross reference and much much more!

Still only $67.50 + tax ($1.50 s/h).

C.C.SOFTWARE
1907 ALVARADO AVE.
WALNUT CREEK, CA 94596
(415)939-8153

Circle 132 on Reader Service Card

SEP/OCT ISSUE SPECIAL: Overstocked back issues only $4.00* each!

Do you need issues 2.4, 3.2, 4.6, 5.4, 6.2, 6.6, 7.2, 7.4, 8.1, or 8.2 to complete your library? They'll never be cheaper than this — so now's the time to get 'em! See page 80 for details.

*Plus $1.50 handling charge per order and $2.50 shipping per magazine to Canada or Mexico, $5.00 shipping per magazine to any other foreign destination. Offer expires 10-31-90.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ CODE IS AVAILABLE!

All the source code from the articles in this issue is available on disk from

Programmer's Journal
P.O. Box 70167
Eugene, OR 97401-0110
800-234-0386
503-747-0800

Code disks are $10.00 each. Please see page 80 for ordering information.

PJ Code Diskscriptions are available for $39.95. See page 16 to order.

The code listings are also available for downloading from several Bulletin Board Systems around the country. See page 67 for listings.

Contact this advertiser directly.

TechnoJock SOFTWARE

If you program in Turbo Pascal, use TechnoJock's Turbo Toolkit for menus, windows, input screens, and much more. $49.95 plus shipping & handling.

"With this Toolkit everyone becomes a TechnoJock."
Turbo User Group

TechnoJock Software, Inc.
PO Box 820927
Houston TX 77282
(713) 493-5354

Circle 289 on Reader Service Card

ATTENTION Software Writers!

Is your software innovative, usable and needed? Are your applications practical, thorough and well tested? Is your documentation well organized and understandable? InSync, a division of an established international software distributor, is evaluating new and innovative software to publish and market domestically and internationally. We are seeking new off-the-shelf MS/PC-DOS software applications (no accounting or inventory, please). ORIGINALITY IS A MUST for all submissions.

Please call our Evaluations Department to discuss submission procedures.

InSync soft ware

1-800-783-7053

Circle 46 on Reader Service Card

PC-lint

C Programmers: Locate bugs, glitches, quirks and inconsistancies in your C code. PC-lint supports multiple modules, K&R, ANSI. Specify OS/2 or MS-DOS. Only $139.

Gimpel Software
(215)584-4261
3207 Hogarth Ln
Collegeville PA 19426
30 Day Money-back Guarantee

Contact this advertiser directly.

Be in (real-) time

Don't reinvent the wheel! Get the Synapse Real-Time eXecutive (SynRTX) source code library to write OBJECT-BASED MULTITASKING applications in C or C++.

- Four distinct classes of functionality: Task, Device, Cpu, and Timeout
- Complete scheduling control
- Serious real-time (no use of MS-DOS)
- Sample of real-time applications to get started (included in demo disk)
- Comprehensive user's manual
- Full library source code included
- No royalties

Demo disk & user's manual $75
SynRTX library source code $2475
Demo disk and user's manual are deducted if library source code purchased. Add $5 for US/Canada and $15 for international shipping and handling. All funds must be U.S. funds.

Cnapse Tel: (514)447-7221 Fax: (514)447-7297
11 Laurier, Suite 103, Chambly, Qc, Canada J3L 9Z7

Circle 53 on Reader Service Card

"QUICK & CLEAN" COBOL

The COBOL Report Activator (COBRA):

- Generates efficient, structured COBOL programs
- Is easy to learn and use
- Accepts your COBOL code and COPY members
- Produces audit trail reports
- Up to 5 output files per program
- Provides report options & site defined defaults
- Is available for Mainframe & PC

COBRA
Systems Software
David Olivadoti, P.O. Box 750817
Petaluma, CA 94975
(707) 763-5123

Circle 281 on Reader Service Card

"Unbelievable"

is what our customers say when they see how many computer books we have available. Unlike book "clubs" we have more than just "best sellers" or one publisher's line. We have the most complete selection for both PC's and the Mac. Please call or write today for our latest specials.

COMPUTER BOOKS
PO Box 70746
Sunnyvale, CA 94086
(408) 730-9477
Fridays 11-2 PM PST

Circle 340 on Reader Service Card

CSOURCE APPLICATION GENERATOR

C code WYSIWYG screen, report, & program generator. Library code may be enhanced for custom generation. Supports multi-user development & DOS 3.1/Novell networks. Produces small-sized DATA ENTRY, REPORT, MENU, INDEX REBUILD & FILE RESTRUCTURE programs with multiple screens & files. Very fast development & execution speeds. Requires 256K RAM, DOS 2.1 or later, Faircom Ctree or Novell Btrieve. For Turbo or Microsoft C 5.0. No royalties, ALL generated applications source is supplied.

$395.00 executable, $595.00 full source code. Multi-programmer licenses.

MBAC
13320 Castana Ave.
Downey, CA 90242
213-531-7259

Circle 291 on Reader Service Card

JT FAX DEVELOPERS

TOUCH-TONE DECODER DAUGHTER CARD

MICROSOFT C LANGUAGE SOURCE DESQVIEW DRIVER (For Multiple Board Systems)

FOR MORE INFORMATION, PLEASE CALL OR FAX:

TDS TELE-DATA SYSTEMS

201-752-3133 ■ Fax: 201-752-7887

Circle 4 on Reader Service Card

POWER GRAPHICS PROGRAMMING

If you do EGA/VGA programming, then *Power Garphics Programming* is a "must-have" for your reference library. This Que book is a compilation of Michael Abrash's first four EGA and nine VGA "On Graphics" articles published in *Programmer's Journal*. It contains great theory, invaluable tips, and nuts-and-bolts ASM and C code listings. A companion code disk is available. Order your book and disk today.

P.G.P. Book $24.95*
P.G.P. Code Disk.....$14.95*

*Plus $1.50 handling charge per order. Free shipping anywhere in the U.S. Add'l. shipping charges to Canada, Mexico, and other foreign destinations.

Order your book and disk direct from:
Programmer's Journal
Call toll free with your VISA/MC
800-234-0386
Outside the U.S. call 503-747-0800

Contact this advertiser directly.

Coming Next Issue:
Database Programming — Old and New

November/December 1990 Vol. 8.6
Advertising space reservations are due:
September 7, 1990
Materials are due:
September 14, 1990

Please call 800-234-0386.
Ask for Jay or Robin in the advertising department to reserve your space.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
305	Acme Software	72
345	Alphax Systems, Inc.	38
41	Applied Intuition	45
30	Austin Code Works, The	70
51	Az-Tech Software, Inc.	92
14	Blaise Computing, Inc.	21
2	Blaise Computing, Inc.	23
15	Blaise Computing, Inc.	25
244	Block Island	73
11	Blue Jay Systems	93
49	Borland International	1
3	Borland International	C4
288	Burton Systems Software	92
132	C. C. Software	93
321	Catenary Systems	35
335	CDP Consultants	93
53	Cnapse	94
281	Cobra Systems Software	94
39	Coleman Softlabs	5
340	Computer Books	93
*	Computer Professionals' Book Society	39
229	Disk Software, Inc.	38
*	Double R Software	74
353	EMS	92
352	EZX	92
178	Flemming Software	93
261	Genus Microprogramming	35
*	Gimpel Software	94
18	Goin Places Software	54
56	Greenview	89
334	Hamilton Laboratories	74
42	Harvard Softworks	56
348	Information Modes	92
358	Inspired Innovations	43
46	InSync Software	94
50	INTEL	45
317	Island Systems	30
205	*Journal of Object-Oriented Programming*	15
211	Kadak Products, Ltd.	90
295	Knowledge Dynamics Corporation	19
155	Komputerwerk, Inc.	92
63	Laboratory Microsystems, Inc.	83
44	Lahey Computer Systems, Inc.	72
246	Magee Enterprises, Inc.	36
223	Marietta Systems, Inc.	92
291	MBAC	94
64	Microdyne Corporation	64
232	Microsoft Human Resources	12
243	Microsoft Press	10
40	Microsystems Software	82
*	Microway	C3
240	Milton Brown Software	47
121	MISOSYS, Inc.	91
266	Mix Software	69
94	MMC AD Systems	63
312	MTD Software Development Co., Ltd.	92
17	Multiscope, Inc.	81
109	Nanosoft	91
107	National Discount Computers	93
160	Nu-Mega Technologies	3
16	Opt-Tech Data Processing	92
254	Opus Software	26
270	OTG Systems, Inc.	74
208	PC Tech, Inc.	35
36	PC Tech, Inc.	83
6	Periscope Company, Inc., The	17
*	***PJ* BBS Listing**	67
*	***PJ* Code DiskScription**	16
*	***PJ* Mail Order Store**	80
54	Programmer's Corner, The	88
48	Programmer's Warehouse	11
60	QNE International	51
252	Quantasm Corporation	85
147	Quarterdeck Office Systems	C2
296	Que Corporation	37
20	Rational Systems, Inc.	18
59	RJSwantek & Associates	7
210	Ryle Design	93
12	S.L.A.M. Software	93
26	Sage Software, Inc.	8–9
32	Sandpaper Software	34
347	Saucci, Andrew M., Jr.	93
65	Schroff Development	49
269	Sequiter Software, Inc.	92
165	SLR Systems	24
167	Software Development '90-Fall	55
356	Software Projects Group	38
110	Softway, Inc.	33
47	Solution Systems	59
10	South Mountain Software	75
330	Stanwood Associates, Inc.	31
289	TechnoJock Software, Inc.	94
4	Teledata Systems	94
166	Tool Makers, The	67
327	Tools & Techniques, Inc.	26
25	TurboPower Software	57
24	Upper Deck Systems	93
341	V Communications, Inc.	68
125	V Communications, Inc.	79
29	Venn	66
328	Vermont Database Corporation	92
304	Visitech Software	91
320	Wenham Software Co.	92
100	Western Wares	47
329	WHS	93
123	XenoSoft	93
182	Zortech, Inc.	27

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers marked with an asterisk (*) prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors. ***Programmer's Journal*** is not responsible for errors in advertisements.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SPARCS Plot World Takeover

By Hal W. Hardenbergh

ILLUSTRATION BY JIM CARPENTER

I work in the high-end image-processing industry. My employer provides its hardware designers with what it believes are the best software tools for schematic entry and documentation. Because this is a group effort, the software has to run under a multi-user operating system, and these days that means UNIX.

The software my employer selected will run on Sun 3s (680x0-based) or Sun 4s (SPARC-based). One of the company's programmers was the first to upgrade from a 3/60 to a SPARCstation I. He quickly discovered that the SPARC was sometimes faster and sometimes slower than the much older 3/60. He missed the consistency of response the 3/60 had provided. Then he installed another 4 megabytes, bringing the total RAM up to 12 megabytes. The SPARC smoothed out and was consistently faster than the 3/60. I've been using a 12 meg SPARCstation myself for schematic entry and documentation for months now. I found the software transition from Sun 3s to Sun 4s was more or less painless.

About the time that I started working on a SPARC, one of my hardware design colleagues left to join a very well-known, fast-growing company (not Sun) in Mountain View so he could try developing image-processing ASICs (Application Specific Integrated Circuits). This company started out making medium-to-high-end, graphics-based UNIX boxes, but is currently making a serious effort to move down price-wise and compete in the low-end UNIX workstation market. So what did this company place on his desk? A SPARCstation. The necessary ASIC design tools don't run on his company's machines; they run on Sun 4s.

It's commonly believed that SPARC workstations are outshipping all other RISC workstations combined, as measured by unit volume. Guess which UNIX workstation folks are developing software for? In other words, the SPARCstation today is what the IBM PC was in 1981-82. The IBM PC was what everybody bought and developed software for. It wasn't high-tech or particularly fast compared to its contemporaries. In fact, certain aspects of its CPU, which was not proprietary, were quite clumsy. *That's also a marvelously accurate description of today's SPARC workstations.*

Being a low-tech device, the PC was easy to clone, and lots of folks did so. This is now happening in the SPARC market. Despite those clones, IBM has managed to hold onto a substantial portion of the PC market. It's too early to tell what's going to happen to Sun's market share in the next few years.

The "clumsy CPU" problem of the original PC has been fixed in the 486, but the 486 still has a "clumsy" mode, and almost all the software you can buy today, nine years after the PC's introduction, still runs in clumsy mode. I think this will happen to the SPARC market as well, for the same reasons. The software developers will want the broadest market, which will include the older (now current) SPARC CPUs. Subsequent SPARC CPUs will have to include a clumsy mode.

Despite the shortcomings of the SPARC CPU architecture, Sun has certainly been successful with its SPARC program. How successful? Well, one measure of success is profits, and Sun has had some problems in that area. Sun's most recent audited quarter (June '89) showed $29 million of red ink. For calendar year '89, Apple did $454 million in profits and Compaq $333 million. Sun made only $61 million.

Sun was not even the most profitable among the companies whose revenues depend mostly on shipping RISC workstations. Intergraph, which ships workstations based on the old Fairchild Clipper, turned a 1989 profit of $80 million. (Profit figures are from the 23 April '90 issue of *Fortune* magazine.)

Nobody can deny that Sun has done a great job of garnering RISC workstation market share as measured by unit volume. To get a fair approximation of the number of computers Sun sold in the quarter, take some recent quarterly results (June '90), and divide the total sales by $21,000. That $21,000 per unit figure is a pretty good estimate of what a system with eight cheapie workstations plus the necessary file server, which is not cheap, will cost. Yep, nearly $200,000. And that's just for the hardware. If you have to ask what the application software costs, you can't afford it.

SPARC workstations are showing up on software and hardware engineers' desks in places where development is done in a group environment and where the eventual application will be running under UNIX. You don't find SPARCs on the desks of secretaries, or middle managers, or in hardware stores or homes, but you do find PCs in those places. Those who think UNIX workstation sales are going to grow without limit apparently don't know that 25% to 40% of all U.S. engineers who work in groups already have one.

In 1989, about 20,000 Cypress SPARC CPUs were sold. Sales were close to 5 million for the 386, and 20 million for the 286. The 486 shipped roughly 40,000 just in the last quarter of 1989. Obviously, 386s and 486s were not gathering dust on any storeroom shelves. Yet some folks seem to think that the SPARC is going to push aside the x86 family. See the last issue of *PJ*, page 15, for instance. (Sorry, Frank.)

What does all this mean? It means that the world's most successful bus manufacturer is never going to outsell automobiles, even though the bus's operating cost is less per passenger mile. Buses are great for group outings where the group has a common goal, but not for much else.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

icroWay manufactures a broad range of products that boost
e speed and capacity of your current PC/AT. They include
6 and 386SX accelerators and 486 replacement mother-
oards. We also offer a complete line of Weitek acces-
ories and stock all of the Intel, Weitek and Cyrix
processors. We created the PC numerics industry in
82 and have been developing, selling and supporting
e best numeric software and hardware ever since.

This XT/AT motherboard
replacement features a 25
MHz 80486, 4167 socket and
a BURST BUS memory inter-
face. The BURST BUS architec-
ture is ideal for engineering,
scientific and CAD/CAM applica-
tions. The NDP Fortran-486 driven
numeric throughput of the 4167 is an
impressive 13.0 Megawhetstones, which
is 100 times the throughput of an 80287
equipped AT!

his AT accelerator board replaces your 80286 with
n 80386 clocked at 20 or 25 MHz. It is socketed for 8
Megabytes of 32 bit RAM, an 80387, Cyrix CX83D87, or
Weitek 3167 and a 64K SRAM cache. The numeric perfor-
nance of the Number Smasher 386/25 is a strong function of
our application and the coprocessor you choose. The 25
MHz NDP Fortran-386 driven Whetstones are 2.1, 3.7 and 5.5
MegaWhetstones running on the 80387, CX83D87 and 3167.

	486	4167
Megawhetstones	5.9	13.0
Megawhetscales	4.1	9.9

Our MCA Weitek card runs in the IBM Model
70 and 80. At 20 MHz, its performance is 2
to 3 times that of an 80387.

NDP Fortran-486 and C-486 are globally optimized main-frame compilers that have been fine tuned for the 80486 and 4167. NDP Fortran-i860 and C-i860 are available in August.

icroWay and Number Smasher are registered trademarks of MicroWay, Inc., 80386, 80387, 80486
re trademarks of Intel Corp., Cyrix and CX83D87 are trademarks of Cyrix Corp., Weitek, 3167 and
167 are trademarks of Weitek Corp.

Coming in August: Number Smasher® i860

World Leader in PC Numerics

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Corporate Headquarters, P.O. Box 79, Kingston, MA 02364 USA (508) 746-7341

New Turbo C++ Professional

When the object is programming

C++

Be Objective

Object-Oriented Programming (OOP) *is* programming in the '90s. It's the next step after structured programming and is the best way to write applications.

And Turbo C++ Professional is the first turbo-charged native code C++ compiler that brings Object-Oriented Programming to your PC.

+ ANSI C

Turbo C++ Professional also compiles ANSI C code, so you can stay productive with C now, and move to C++ at your own pace.

Environment ++

The best compiler deserves the best environment, and our new Programmer's Platform™ environment makes you more productive. It features overlapping windows and mouse support. And sports a new multi-file editor, an integrated debugger, and a smart project manager. Its advanced open architecture lets you integrate the tools you need to feel right at home.

VROOMM adds room

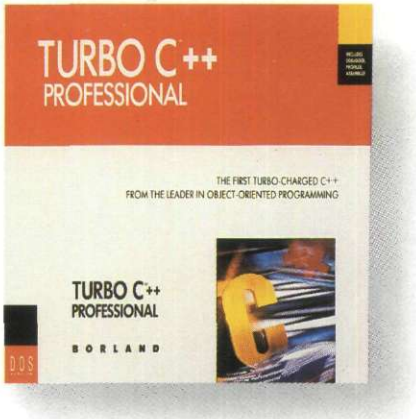
VROOMM™ (Virtual Runtime Object-Oriented Memory Manager) lets you break the 640K barrier. Just select the application code you want to overlay, and VROOMM does the rest—swapping modules on demand. It's fast, easy, automatic.

Another +

Turbo C++ Professional gives you all the tools you need to build fast, reliable C++ programs.

Turbo Debugger® 2.0 debugs your object-oriented programs. This powerful new version is the first and only debugger to support *reverse execution*. Letting you step backwards through your code to find the bugs you might have missed.

New Turbo Profiler,™ the world's first interactive profiler, displays histograms of your program's performance. With it, you can easily spot execution bottlenecks, and see where improvements or redesign of your code will yield maximum performance gains.



And Turbo Assembler® 2.0 lets you replace time-critical segments of your code using the world's fastest MASM-compatible assembler.

Turbo C++ Professional Compiler

- C++ conforming to AT&T's 2.0 specification
- C++ class libraries
- Full ANSI C compiler
- VROOMM overlay manager
- Complete documentation and tutorials

Programmer's Platform

- Open architecture for integration of your own tools
- Overlapping windows with mouse support
- Multifile, macro-based editor
- Smart project manager provides visual MAKE
- Integrated debugging and hypertext help

Turbo Debugger 2.0

- Class hierarchy browser and inspectors
- Reverse execution provides "true" undo
- 286 protected-mode and 386 virtual-mode debugging
- Keystroke record and playback

NEW Turbo Profiler

- Displays histograms of program execution
- Tracks call history, overlays, interrupts, file I/O

Turbo Assembler 2.0

- Multipass assembler with NOP squishing and 486 support

Special Offer

The suggested retail price for Turbo C++ Professional is $299.95 ($199.95 for Turbo C++). Borland is offering a special discount for registered Turbo C owners. So be objective, and **SEE YOUR DEALER** or call Borland at **1-800-331-0877** now!

BORLAND

Code: MC87

Mail orders to: Borland, P.O. Box 660001, Scotts Valley, CA 95067-0001. For orders outside the U.S., call (408) 438-5300.
Turbo C, VROOMM, Turbo Debugger, Turbo Profiler and Turbo Assembler are trademarks or registered trademarks of Borland International, Inc. Copyright © 1990, Borland International, Inc. All rights reserved. BI-1333B

Circle 3 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED